IBM WebSphere Real Time for Linux
Version 3

Guide d'utilisation

IBM WebSphere Real Time for Linux
Version 3

Guide d'utilisation

Important

Avant d'utiliser le présent document et le produit associé, prenez connaissance des informations générales figurant à la section Chapitre 11, «Remarques», à la page 69.

Première édition (août 2011)

Cette édition du guide d'utilisation s'applique à IBM WebSphere Real Time for Linux, Version 3 et à toutes les éditions et modifications suivantes, sauf indication contraire dans les nouvelles éditions.

Table des matières

Figures

Tableaux

Préface

Ce guide d'utilisation fournit des informations générales sur IBM® WebSphere Real Time for Linux.

Chapitre 1. Introduction

Cette section fournit des informations sur IBM WebSphere Real Time for Linux.

Présentation de WebSphere Real Time for Linux

WebSphere Real Time for Linux intègre les fonctionnalités temps réel avec la machine virtuelle (JVM) J9 IBM.

WebSphere Real Time for Linux est un produit Java Runtime Environment doté d'un kit de développement de logiciels (SDK) qui apporte à IBM SDK for Java les fonctionnalités temps réel. Les applications qui dépendent de temps de réponse précis peuvent tirer avantage des fonctionnalités temps réel fournies avec WebSphere Real Time for Linux avec la technologie Java standard.

Fonctions

Les applications temps réel ont besoin d'exécution cohérente et non pas d'une vitesse absolue.

Les principaux inconvénients du déploiement d'applications temps réel avec des machines JVM classiques sont les suivants :
- Délais imprévisibles (potentiellement longs) du récupérateur de place.
- Retards d'exécution des méthodes lors de la compilation et de la recompilation JIT (Just-In-Time) avec une durée d'exécution variable.
- Planification arbitraire du système d'exploitation.

WebSphere Real Time for Linux élimine ces obstacles en fournissant :
- Le récupérateur de place Metronome est une fonction déterministe incrémentielle avec des délais d'interruption très courts.

Nouveautés

Cette rubrique présente les modifications relatives à IBM WebSphere Real Time for Linux.

WebSphere Real Time for Linux V3

WebSphere Real Time for Linux V3 est une extension d'IBM SDK for Java 7, fondée sur les fonctions disponibles dans cette version afin d'inclure les fonctionnalités temps réel. Les versions antérieures de WebSphere Real Time for Linux étaient basées sur des versions antérieures d'IBM SDK for Java.

Pour en savoir plus sur les nouveautés, voir : What's new dans le centre de documentation d'IBM SDK for Java 7.

Contrôle des délais d'interruption pour le récupérateur de place Metronome

Par défaut, le délai d'interruption du récupérateur de place Metronome entre les cycles de récupération est égal à 3 millisecondes. Vous pouvez modifier cette valeur pour contrôler le délai d'interruption à l'aide d'une nouvelle option de ligne de commande. Pour plus d'informations sur cette option, voir «Contrôle du délai d'interruption», à la page 15.

Références compressées

Le récupérateur de place Metronome prend maintenant en charge les références non compressées ainsi que les références compressées sur les plateformes 64 bits. Pour plus d'informations sur l'impact sur les performances, voir Chapitre 7, «Performances», à la page 23.

Avantages

L'environnement a pour avantages de permettre aux applications Java de s'exécuter avec un meilleur niveau de prévisibilité qu'avec la machine JVM standard et de fournir un timing cohérent aux applications Java. Les activités en arrière-plan, telles que la compilation et la récupération de place, sont exécutées à certains moments, ce qui élimine les pics imprévus d'activité en arrière-plan lors de l'exécution de l'application.

Pour tirer parti de ces avantages, étendez la machine JVM avec la technologie de récupération de place temps réel Metronome.

Chapitre 2. Description d'IBM WebSphere Real Time for Linux

Cette section présente les composants clés d'IBM WebSphere Real Time for Linux.

- «Présentation du récupérateur de place Metronome»

Présentation du récupérateur de place Metronome

Le récupérateur de place Metronome remplace le récupérateur de place standard dans WebSphere Real Time for Linux.

La principale différence entre la récupération de place Metronome et la récupération de place standard réside dans le fait que la récupération de place Metronome intervient par petites étapes interruptibles alors que la récupération de place standard arrête l'application lorsqu'elle marque et récupère la place.

Par exemple :

```
java -Xgcpolicy:metronome -Xgc:targetUtilization=80 yourApplication
```

L'exemple spécifie que l'application s'exécute à 80 % toutes les 60 ms. Les 20 % du temps restant peuvent être utilisés pour la récupération de place. Le récupérateur de place Metronome garantit des niveaux d'utilisation dès lors qu'il dispose des ressources suffisantes. La récupération de place commence lorsque la quantité d'espace libre dans le segment de mémoire tombe en dessous d'un seuil défini dynamiquement.

Récupération de place Metronome et déchargement de classe

Metronome prend en charge le déchargement de classe de la même manière qu'un kit de développement standard Java. Toutefois, compte tenu du travail associé, il peut exister des dépassements de durée de pause au cours de la récupération de place lors du déchargement des classes.

Unités d'exécution du récupérateur de place Metronome

Le récupérateur de place Metronome est constitué de deux types d'unités d'exécution : une unité d'exécution d'alarme et un certain nombre d'unités d'exécution de récupération de place. Par défaut, la récupération de place utilise une unité d'exécution pour chaque processeur actif logique disponible pour le système d'exploitation, ce qui permet d'exécuter un traitement parallèle optimal au cours des cycles de récupération de place. Un cycle de récupération de place correspond au délai entre le déclenchement de la récupération de place et la fin de la libération de la place. Selon la taille de segment de mémoire Java, un cycle de récupération de place peut durer plusieurs secondes. Un cycle de récupération de place contient généralement des centaines de quanta de récupération de place. Ces quanta sont de très petites pauses dans le code d'application qui durent généralement 3 millisecondes. Utilisez **-verbose:gc** pour générer des résumés sur les cycles et les quantas. Pour plus d'informations, voir «Utilisation des informations verbose:gc», à la page 56. Vous pouvez définir le nombre d'unités d'exécution de récupération de place de la machine JVM en utilisant l'option **-Xgcthreads**.

L'augmentation de la valeur par défaut **-Xgcthreads** n'offre aucun avantage. La réduction de la valeur **-Xgcthreads** peut réduire la charge UC globale pendant les cycles de récupération de place, mais ces derniers seront plus longs.

Remarque : Les cibles de durée des quantas de récupération de place restent constantes à 3 millisecondes.

Vous ne pouvez pas changer le nombre d'unités d'exécution d'alarme de la machine JVM.

Le récupérateur de place Metronome vérifie régulièrement la machine JVM pour déterminer si le segment de mémoire contient un espace libre suffisant. Lorsque la quantité d'espace libre tombe en dessous de la limite, le récupérateur de place Metronome déclenche la machine JVM pour lancer la récupération de place.

Unité d'exécution d'alarme

La seule unité d'exécution d'alarme permet d'utiliser une quantité minimale de ressources. Elle «s'active» régulièrement et elle vérifie :

- la quantité d'espace libre dans le segment de mémoire
- si la récupération de place est active.

Si l'espace est insuffisant et que la récupération de place est en cours, l'unité d'exécution d'alarme déclenche les unités d'exécution de récupération pour lancer la récupération de place. L'unité d'exécution d'alarme ne fait rien jusqu'à la prochaine vérification JVM planifiée.

Unités d'exécution de récupération

Les unités d'exécution de récupération récupèrent la place.

A la fin du cycle de récupération de place, le récupérateur de place Metronome vérifie la quantité d'espace libre dans le segment de mémoire. Si l'espace est toujours insuffisant, un nouveau cycle de récupération de place démarre en utilisant le même ID de déclencheur. Si l'espace est suffisant, le déclencheur et les unités d'exécution de récupération de place s'arrêtent. L'unité d'exécution d'alarme continue de contrôler l'espace libre dans le segment de mémoire et déclenche un autre cycle de récupération de place lorsque cela est nécessaire.

Pour plus d'informations sur l'utilisation du récupérateur de place Metronome, voir «Utilisation du récupérateur de place Metronome», à la page 15.

Chapitre 3. Planification

Lisez cette section avant d'installer WebSphere Real Time for Linux.

-
- «Environnements pris en charge»
-
- «Considérations»

Migration

Vous pouvez exécuter vos applications Java standard WebSphere Real Time for Linux sans modification.

Environnements pris en charge

IBM WebSphere Real Time for Linux est pris en charge sur certaines plateformes matérielles et certains systèmes d'exploitation.

IBM WebSphere Real Time for Linux

Les architectures de plateforme suivantes sont prises en charge :

- Intel Architecture, 32 bits (IA-32)
  - Pentium 4
  - Pentium Xeon
  - Pentium M
  - Pentium D et équivalents
- AMD64/EM64T
- IBM POWER 32
- IBM POWER 64

Remarque : Le matériel Pentium 3 n'est plus pris en charge.

Les systèmes d'exploitation suivants sont pris en charge :

Tableau 1. Environnements Linux testés

Matériel	**IA-32 32 bits**	**AMD64/EM64T 64 bits**	
Espaces adresse SDK	**32 bits**	**32 bits**	**64 bits**
RHEL 5 Mise à jour 7	Oui	Oui	Oui
RHEL 6 Mise à jour 1	Oui	Oui	Oui
SLES 11 SP1	Oui	Oui	Oui

Remarque : SLES 9, SLES 10 et RHEL 4 ne sont pas pris en charge.

Considérations

Vous devez tenir compte d'un certain nombre de points lors de l'utilisation de WebSphere Real Time for Linux.

- Dans la mesure du possible, exécutez une seule JVM temps réel sur un même système pour ne pas avoir plusieurs récupérateurs de place. Chaque machine JVM n'a pas connaissance des zones de mémoire de l'autre JVM. Une conséquence est que les cycles de récupération de place et les délais d'interruption ne peuvent pas être coordonnés sur les machines JVM, ce qui signifie qu'une machine JVM peut affecter négativement les performances du récupérateur de place sur une autre machine JVM. Si vous devez utiliser plusieurs machines JVM, assurez-vous que chacune d'elles est liée à un sous-ensemble spécifique de processeurs en lançant la commande **taskset**.
- Les caches partagés utilisés par les versions antérieures de WebSphere Real Time for Linux pour stocker le code précompilé et les classes ne sont pas compatibles avec ceux utilisés par cette version du produit. Vous devez régénérer le contenu des caches précédents.
- Lorsque vous utilisez des caches de classes partagées, le nom du cache ne doit pas dépasser 53 caractères.

Chapitre 4. Installation de WebSphere Real Time for Linux

Pour installer le produit, procédez comme suit.

Fichiers d'installation

Les fichiers d'installation nécessaires sont les suivants.

IBM WebSphere Real Time for Linux est fourni dans deux types de package InstallAnywhere.

Packages installables

Les packages installables permettent de configurer votre systèmes. Par exemple, les programmes peuvent définir des variables d'environnement.

- `wrt-3.0-0.0-linux-<arch>-sdk.bin`
- `wrt-3.0-0.0-linux-<arch>-jre.bin`

Packages d'archivage

Ces packages extraient les fichiers sur votre système mais n'effectuent aucune configuration.

- `wrt-3.0-0.0-linux-<arch>-sdk-archive.bin`
- `wrt-3.0-0.0-linux-<arch>-jre-archive.bin`

Remarque : <arch> est votre architecture de plateforme ; x86_32 ou x86_64.

Installation à partir d'un package InstallAnywhere

Ces packages fournissent un programme interactif qui vous guide tout au long du choix des options d'installation. Vous pouvez exécuter ce programme sous forme d'interface graphique ou à partir d'une console système.

Avant de commencer

Votre système doit disposer des deux bibliothèques partagées suivantes :

- Bibliothèque C GNU V2.3 (glibc)
- `libstdc++.so.5`

Si vous ne disposez pas de la bibliothèque partagée `libstdc++.so.5`, il se peut que lors de l'installation s'affiche le cliché Javacore, contenant les erreurs suivantes :

```
JVMJ9VM011W Unable to load j9dmp24: libstdc++.so.5: cannot open shared object file:
No such file or directory
JVMJ9VM011W Unable to load j9gc24: libstdc++.so.5: cannot open shared object file:
No such file or directory
JVMJ9VM011W Unable to load j9vrb24: libstdc++.so.5: cannot open shared object file:
No such file or directory
```

Si vous installez un package installable, l'outil m-build doit être installé sur votre système. Dans le cas contraire, le programme d'installation ne peut pas enregistrer le nouveau package dans la base de données RPM. Pour savoir si l'outil rpm-build est installé, entrez la commande suivante :

```
rpm -q rpm-build
```

Pourquoi et quand exécuter cette tâche

Les packages InstallAnywhere possèdent une extension de fichier `.bin`.

Il existe deux types de package :

Installable
L'installation de ces packages configure également votre système (par exemple, en définissant les variables d'environnement).

Archive
L'installation de ces packages extrait les fichiers sur votre système, mais n'effectue aucune configuration.

Procédure

- Pour installer le package de façon interactive, effectuez une installation avec opérateur.
- Pour installer le package sans aucune interaction supplémentaire avec l'utilisateur, effectuez une installation automatique. Vous pouvez choisir cette option si vous effectuez l'installation sur de nombreux système.
- A la fin de l'installation, suivez la procédure d'installation fournie dans cette section, notamment pour définir les variables d'environnement path et classpath.

Résultats

Le produit est installé.

Remarque : N'interrompez pas le processus d'installation, par exemple en appuyant sur Ctrl+C. Si vous interrompez le processus, il pourra être nécessaire de réinstaller le produit. Pour plus d'informations, voir «Installation interrompue», à la page 11.

Si vous utilisez un package installable, des messages vous indiquant la survenue d'un problème peuvent s'afficher. Par contre, l'installation des packages d'archives ne génère aucun message. Certains des messages pouvant s'afficher lors de l'utilisation d'un package installable figurent dans la liste ci-après :

`Le programme d'installation ne peut pas être exécuté dans votre configuration. Il va se fermer.`
Ce message d'erreur s'affiche lorsque votre ID utilisateur n'est pas autorisé à exécuter le processus d'installation. Etant donné que l'installation ne peut pas se poursuivre, le programme se ferme. Pour résoudre ce problème, relancez l'installation avec un ID utilisateur ayant des droits d'accès root.

Un package RPM est déjà installé. Désinstallez ce package avant de continuer.

Ce message indique qu'un package RPM est déjà installé. Etant donné que l'installation ne peut pas se poursuivre, le programme se ferme. Pour résoudre ce problème, désinstallez le package RPM avant de continuer.

Exécution d'une installation avec opérateur

Installez le produit à partir d'un module InstallAnywhere de façon interactive.

Avant de commencer

Avant de commencer l'installation, vérifiez que les conditions suivantes sont réunies :

- Si vous avez déjà installé WebSphere Real Time for Linux à partir d'un package RPM, vous devez désinstaller ce dernier avant de continuer.
- Vous devez disposer d'un ID utilisateur avec des droits d'accès root.

Procédure

1. Téléchargez le fichier du package d'installation dans un répertoire temporaire.
2. Placez-vous dans le répertoire temporaire.
3. Démarrez la procédure d'installation en entrant `./`*package* à l'invite shell où *package* est le nom du package que vous installez.
4. Sélectionnez une langue dans la liste qui s'affiche dans la fenêtre du programme d'installation, puis cliquez sur **Next (Suivant)**. La liste des langues disponibles dépend des paramètres régionaux définis pour votre système.
5. Lisez le contrat de licence, en utilisant la barre de défilement pour accéder à la fin du texte de licence. Pour pouvoir effectuer l'installation, vous devez accepter les conditions du contrat de licence. Pour accepter les conditions du contrat, sélectionnez le bouton d'option correspondant et cliquez sur **OK**.

   Remarque : Vous ne pouvez pas sélectionner le bouton d'option pour accepter le contrat de licence tant que vous n'avez pas lu le texte de licence jusqu'à la fin.
6. Le système vous invite à choisir le répertoire cible pour l'installation. Si vous ne souhaitez pas effectuer l'installation dans le répertoire par défaut, cliquez sur **Choose (Sélection)** pour sélectionner un autre répertoire à l'aide de la fenêtre de navigation. Une fois le répertoire d'installation sélectionné, cliquez sur **Next (Suivant)** pour continuer.
7. Le système vous invite à passer en revue les choix que vous avez effectués. Pour modifier votre sélection, cliquez sur **Previous (Précédent)**. Si vos choix sont corrects, cliquez sur **Install (Installer)** pour procéder à l'installation.
8. Une fois l'installation terminée, cliquez sur **Done (Terminé)** pour terminer.

Exécution d'une installation automatique

Si vous devez effectuer l'installation sur plusieurs systèmes et que vous connaissez déjà les options d'installation que vous souhaitez utiliser, il peut être intéressant d'utiliser le processus d'installation automatique. Vous effectuez l'installation une fois à l'aide du processus d'installation avec opérateur, puis vous utilisez le fichier de réponses résultant pour effectuer les autres installations sans interaction supplémentaire avec l'utilisateur.

Procédure

1. Créez un fichier de réponses en effectuant une installation avec opérateur. Utilisez l'une des solutions suivantes :
   - Spécifiez à partir de l'interface graphique que le programme d'installation devra créer un fichier de réponses. Le fichier de réponses s'appelle `installer.properties` et est créé dans le répertoire d'installation.
   - A partir de la ligne de commande, ajoutez l'option -r à la commande d'installation avec opérateur, en spécifiant le chemin d'accès complet au fichier de réponses. Par exemple :

     `./package -r /path/installer.properties`

   Exemple de contenu de fichier de réponses :

   ```
   INSTALLER_UI=silent
   USER_INSTALL_DIR=/mon_répertoire
   ```

   Dans cet exemple, */mon_répertoire* représente le répertoire d'installation cible que vous avez choisi pour l'installation.
2. Facultatif : Le cas échéant, modifiez les options dans le fichier de réponses.

   Remarque : Les packages d'archives présentent le problème connu suivant : les installations qui utilisent un fichier de réponses utilisent le répertoire par défaut même si vous avez modifié ce répertoire dans le fichier de réponses. S'il existe une installation précédente dans le répertoire par défaut, elle est remplacée.

   Si vous créez plusieurs fichiers de réponses, chacun contenant des options d'installation différentes, spécifiez un nom unique pour chaque fichier de réponses, au format *monfichier*`.properties`.
3. Facultatif : Générez un fichier journal. Etant donné que vous effectuez une installation automatique, aucun message d'état n'est affiché à la fin de l'installation. Pour générer un fichier journal contenant l'état de l'installation, procédez comme suit :
   a. Définissez les propriétés système requises à l'aide de la commande suivante :

      ```
      export _JAVA_OPTIONS="-Dlax.debug.level=3 -Dlax.debug.all=true"
      ```
   b. Définissez la variable d'environnement suivante pour envoyer la sortie du journal sur la console.

      ```
      export LAX_DEBUG=1
      ```
4. Lancez une installation automatique en exécutant le programme d'installation du package à l'aide de l'option **-i** silent et de l'option **-f** pour spécifier le fichier de réponses. Par exemple :

   ```
   ./package -i silent -f
   /chemin/installer.properties 1>console.txt 2>&1
   ```

   ```
   ./package -i silent -f
   /chemin/monfichier.properties 1>console.txt 2>&1
   ```

   Vous pouvez utiliser un chemin d'accès complet ou relatif au fichier de propriétés. Dans ces exemples, la chaîne `1>console.txt 2>&1` redirige les informations du processus d'installation des flux stderr et stdout vers le fichier journal `console.txt` dans le répertoire en cours. Prenez connaissance du fichier journal si vous pensez qu'il y a eu un problème durant l'installation.

   Remarque : Si le répertoire d'installation contient plusieurs fichiers de réponses, c'est le fichier de réponses par défaut, `installer.properties` qui est utilisé.

Installation interrompue

Si le programme d'installation du package est interrompu de façon inattendue durant l'installation (par exemple, si vous appuyez sur Ctrl+C), l'installation est endommagée et vous ne pouvez pas désinstaller ou réinstaller le produit. Si vous tentez d'effectuer une désinstallation ou une réinstallation, le message `Fatal Application Error` s'affiche.

Pourquoi et quand exécuter cette tâche

Pour résoudre ce problème, supprimez les fichiers et réinstallez en suivant les étapes indiquées ci-après.

Procédure

1. Supprimez le fichier de registre `/var/.com.zerog.registry.xml`.
2. Supprimez le répertoire contenant l'installation s'il a été créé. Par exemple, /opt/IBM/javawrt3[_64]/.
3. Relancez le programme d'installation.

Problèmes connus et limitations

Les packages InstallAnywhere présentent certains problèmes connus ainsi que certaines limitations.

- Si vous ne disposez pas de la bibliothèque partagée `libstdc++.so.5` sur votre système, l'installation échoue et génère un cliché Javacore. Pour plus d'informations, voir «Installation à partir d'un package InstallAnywhere», à la page 7.
- L'interface graphique du package d'installation ne prend pas en charge le programme de lecture d'écran Orca. Vous pouvez utiliser le mode installation automatique à la place de l'interface graphique.
- Si après l'installation vous entrez **./*package*** pour redémarrer le programme, ce dernier affiche le message suivant :

  ```
  ENTREZ LE NUMERO DE VOTRE CHOIX OU APPUYEZ SUR <ENTREE> POUR
  ACCEPTER LA VALEUR PAR DEFAUT :
  ```

  Si vous appuyez sur Entrée pour accepter la valeur par défaut, le programme ne répond pas. Entrez un nombre, puis appuyez sur Entrée.
- Si vous installez le package, puis tentez à nouveau d'effectuer l'installation dans un mode différent (par exemple, mode console ou installation automatique), le message suivant peut s'afficher :

  ```
  Invocation of this Java Application has caused an InvocationTargetException.
  This application will now exit
  ```

  Vous ne devriez pas voir ce message s'afficher si vous avez effectué l'installation en mode interface graphique et que vous exécutez à nouveau le programme d'installation en mode console. Si cette erreur s'affiche et que vous exécutez le programme pour sélectionner l'option de désinstallation (packages installables uniquement), utilisez plutôt la commande **./_uninstall/uninstall** comme indiqué dans la section «Désinstallation de WebSphere Real Time for Linux», à la page 14.

Packages installables uniquement

- Vous ne pouvez pas mettre à niveau une installation existante à l'aide des packages InstallAnywhere. Pour effectuer la mise à niveau d'WebSphere Real Time for Linux, vous devez commencer par désinstaller les précédentes versions.

- Il n'est pas possible d'installer deux instances différentes de la même version de WebSphere Real Time for Linux sur un même système en même temps, même si vous utilisez des répertoires d'installation différents. Par exemple, vous ne pouvez pas avoir simultanément WebSphere Real Time for Linux V3 dans le répertoire `/previous` et une installation d'actualisation de service WebSphere Real Time for Linux dans le répertoire `/current`. Le programme d'installation vérifie le numéro de version. S'il trouve un package existant avec le même numéro de version, vous êtes invité à désinstaller le package existant.
- Si le package est installé et que vous exécutez à nouveau le programme d'installation de package à l'aide de l'interface graphique, vous pouvez sélectionner la désinstallation du package. Cette option de désinstallation n'est pas disponible en mode automatique. Si vous réexécutez le programme d'installation du package en mode automatique, le programme s'exécute mais n'effectue aucune action.

Packages d'archives uniquement

- Si vous modifiez le répertoire d'installation dans un fichier de réponses, puis exécutez une installation automatique à l'aide de ce fichier de réponses, le programme d'installation ne tient pas compte du nouveau répertoire d'installation et utilise le répertoire par défaut. S'il existe une installation précédente dans le répertoire par défaut, elle est remplacée.

Définition du PATH

Après avoir défini la variable d'environnement **PATH**, vous pouvez exécuter une application ou un programme en tapant son nom à partir d'une invite shell.

Pourquoi et quand exécuter cette tâche

Remarque : Si vous modifiez la variable d'environnement **PATH** comme indiqué dans la section, vous remplacez les exécutables Java qui existent dans le chemin.

Vous pouvez définir le chemin d'accès à un outil en tapant le chemin chaque fois avant le nom de l'outil. Par exemple, si le kit SDK est installé dans `/opt/IBM/javawrt3[_64]/`, vous pouvez compiler le fichier *monfichier.java* en tapant la commande suivante à partir d'une invite shell :

```
/opt/IBM/javawrt3[_64]/bin/javac myfile.java
```

Pour éviter d'entrer le chemin complet à chaque fois :

1. Editez le fichier de démarrage du shell dans votre répertoire initial (généralement **.bashrc**, en fonction du shell) et ajoutez les chemins absolus à la variable d'environnement **PATH**. Par exemple :

   ```
   export PATH=/opt/IBM/javawrt3[_64]/bin:/opt/IBM/javawrt3[_64]/jre/bin:$PATH
   ```
2. Connectez-vous de nouveau ou exécutez le script de shell mis à jour pour activer le nouveau paramètre **PATH**.
3. Compilez le fichier à l'aide de l'outil **javac**. Par exemple, pour compiler le fichier *monfichier.java*, entrez ce qui suit à l'invite shell :

   ```
   javac -Xgcpolicy:metronome monfichier.java
   ```

   La variable d'environnement **PATH** permet à Linux de localiser les fichiers exécutables, tels que **javac**, **java** et **javadoc**, à partir du répertoire en cours. Pour afficher la valeur actuelle du chemin, entrez ce qui suit depuis une invite de commande :

   ```
   echo $PATH
   ```

Que faire ensuite

Voir «Définition du chemin d'accès aux classes» pour savoir si vous devez définir la variable d'environnement CLASSPATH.

Définition du chemin d'accès aux classes

La variable d'environnement **CLASSPATH** indique aux outils SDK, **java**, **javac**, et **javadoc**, où trouver les bibliothèques de classe Java.

Pourquoi et quand exécuter cette tâche

Définissez la variable d'environnement **CLASSPATH** explicitement uniquement dans les conditions suivantes :

- Vous avez besoin d'une bibliothèque ou d'un fichier de classes différent, que vous pouvez développer par exemple, et elle/il ne se trouve pas dans le répertoire de travail.
- Vous changez l'emplacement des répertoires `bin` et `lib` et ils n'ont plus le même répertoire parent.
- Vous prévoyez de développer ou d'exécuter des applications qui utilisent différents environnements d'exécution sur le même système.

Pour afficher la valeur en cours **CLASSPATH**, entrez les informations suivantes depuis une invite shell :

```
echo $CLASSPATH
```

Si vous développez et exécutez des applications qui utilisent différents environnements d'exécution, y compris d'autres versions que vous avez installées séparément, vous devez définir **CLASSPATH** et **PATH** explicitement pour chaque application. Si vous exécutez plusieurs applications simultanément et utilisez des environnements d'exécution différents, chaque application doit être exécutée dans son propre shell.

Si vous exécutez une seule version de Java à la fois, vous pouvez utiliser un script shell pour passer d'un environnement à l'autre.

Que faire ensuite

Voir «Test de l'installation» pour vérifier que l'installation a abouti.

Test de l'installation

Utilisez l'option **-version** pour déterminer si l'installation est correcte.

Pourquoi et quand exécuter cette tâche

L'installation Java est constituée d'une machine JVM temps réel.

Procédure

Testez l'application comme suit :

1. Pour afficher les informations des versions de la machine JVM temps réel, tapez la commande suivante depuis une invite shell :

   ```
   java -Xgcpolicy:metronome -version
   ```

Cette commande retourne les messages suivants si elle aboutit :

```
java version "1.7.0"
WebSphere Real Time V3(build pxi3270-20110428_04)
IBM J9 VM (build 2.6, JRE 1.7.0 Linux x86-32 20110427_81014 (JIT enabled, AOT
enabled)
J9VM - R26_head_20110426_2022_B81001
JIT  - r11_20110426_19388
GC   - R26_head_20110426_1548_B80973
J9CL - 20110427_81014)
JCL - 20110427_03 based on Oracle 7b145
```

Remarque : Les informations de version sont correctes mais l'architecture de plateforme et les dates peuvent différer ce celles fournies dans l'exemple. Le format de la chaîne de date est `aaaammjj` suivi éventuellement d'informations supplémentaires spécifiques du composant.

Désinstallation de WebSphere Real Time for Linux

La procédure utilisée pour supprimer WebSphere Real Time for Linux dépend du type d'installation utilisé.

Avant de commencer

Concernant les packages installables InstallAnywhere, vous devez disposer d'un ID utilisateur avec des droits d'accès root.

Pourquoi et quand exécuter cette tâche

Il n'existe aucune procédure de désinstallation pour les packages d'archives InstallAnywhere. Pour supprimer un package d'archives de votre système, supprimez le répertoire cible que vous avez sélectionné lors de l'installation du package. Concernant les packages installables InstallAnywhere, vous pouvez désinstaller le produit à l'aide d'une commande ou en exécutant à nouveau le programme d'installation, comme indiqué dans les étapes ci-après.

Procédure

- Facultatif : Désinstallez manuellement à l'aide de la commande **uninstall**.
  1. Accédez au répertoire contenant l'installation. Par exemple :
     ```
     cd /opt/IBM/javawrt3
     ```
  2. Lancez la procédure de désinstallation en entrant la commande suivante :
     ```
     ./_uninstall/uninstall
     ```
- Facultatif : Si vous ne parvenez pas à localiser facilement le programme de désinstallation, vous pouvez aussi exécuter une autre installation avec opérateur. Le programme d'installation détecte alors que le produit est déjà installé, puis vous offre la possibilité de désinstaller l'installation précédente.

Chapitre 5. Exécution des applications IBM WebSphere Real Time for Linux

Ces informations sont très importantes et vous aideront à exécuter les applications en temps réel.

•

• «Utilisation du récupérateur de place Metronome»

Utilisation du récupérateur de place Metronome

Le récupérateur de place Metronome remplace le récupérateur de place standard dans WebSphere Real Time for Linux.

Contrôle du délai d'interruption

Le délai d'interruption du récupérateur de place Metronome peut être réglé pour chaque processus Java.

Par défaut, le récupérateur de place Metronome effectue une pause de 3 millisecondes lors de chaque pause individuelle : cela s'appelle un quantum. Un cycle complet de récupération de place nécessite de nombreuses pauses, ces pauses (ou interruptions) étant réparties afin de laisser à l'application suffisamment de temps pour s'exécuter. Vous pouvez modifier la valeur du délai d'interruption individuelle maximal à l'aide de l'option **-Xgc:targetPauseTime**. Par exemple, si vous définissez **-Xgc:targetPauseTime=20**, le récupérateur de place effectuera des pauses individuelles d'une durée maximale de 20 millisecondes.

Vous pouvez utiliser les outils IBM Monitoring and Diagnostics Tools for Java - Garbage Collection and Memory Visualizer (GCMV) pour contrôler les délais d'interruption du récupérateur de place pour votre application, ainsi que pour diagnostiquer et régler les problèmes de performances de votre application Java. L'outil analyse les données et effectue un tracé correspondant à partir de divers types de journaux, en particulier :

- Les journaux de la récupération de place prolixes
- Les journaux de trace de la récupération de place, générés à l'aide du paramètre **-Xtgc**.
- Les journaux de la mémoire native, générés à l'aide des commandes système **ps**, **svmon** ou **perfmon**.

Les graphiques présentés dans cette section sont générés par GCMV et présentent les effets d'un changement du délai d'interruption cible sur les cycles du récupérateur de place. Chaque graphique présente les délais d'interruption réels entre les cycles de récupération de place Metronome (axe des Y) par rapport au temps d'exécution d'une application (axe des X).

Remarque : GCMV prend en charge un format de récupération de place en mode prolixe antérieur. Si vous voulez analyser la sortie GC prolixe avec GCMV, génère la sortie avec l'option **-Xgc:verboseFormat=deprecated**. Pour plus d'informations, voir Options de ligne de commande GC.

Si l'on définit le délai d'interruption cible par défaut, le graphique des délais d'interruption du récupérateur de place prolixe indique que ces délais ne dépassent

pas 3 millisecondes :

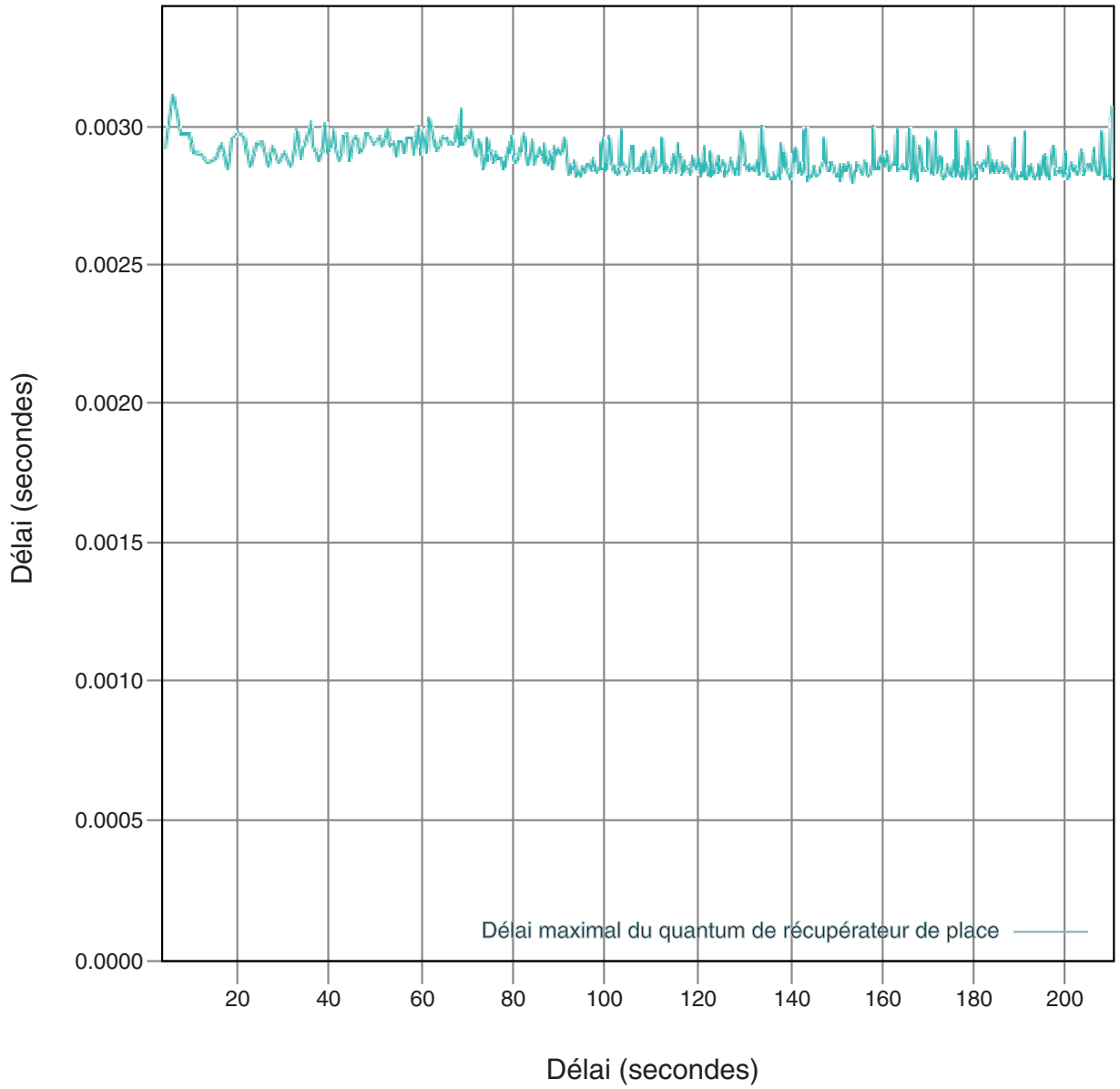

Figure 1. Délais d'interruption réels de la récupération de place lorsque le délai d'interruption cible est défini par sa valeur par défaut (3 millisecondes)

Si l'on définit le délai d'interruption cible à 6 millisecondes, le graphique des délais d'interruption du récupérateur de place prolixe indique que ces délais ne dépassent pas 6 millisecondes :

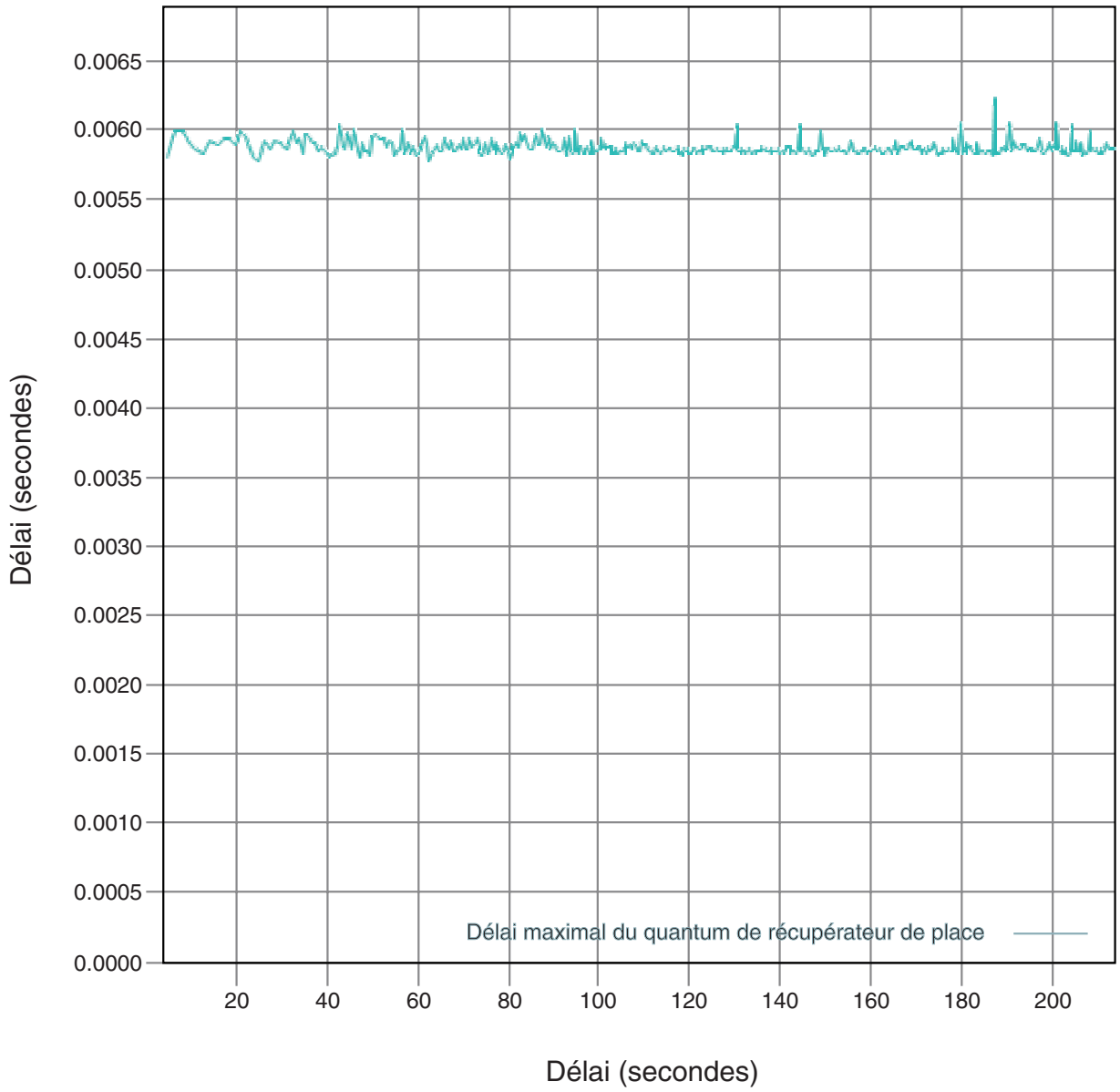

Figure 2. Délais d'interruption réels lorsque le délai d'interruption cible est défini à 6 millisecondes

Si l'on définit le délai d'interruption cible à 10 millisecondes, le graphique des délais d'interruption du récupérateur de place prolixe indique que ces délais ne dépassent pas 10 millisecondes :

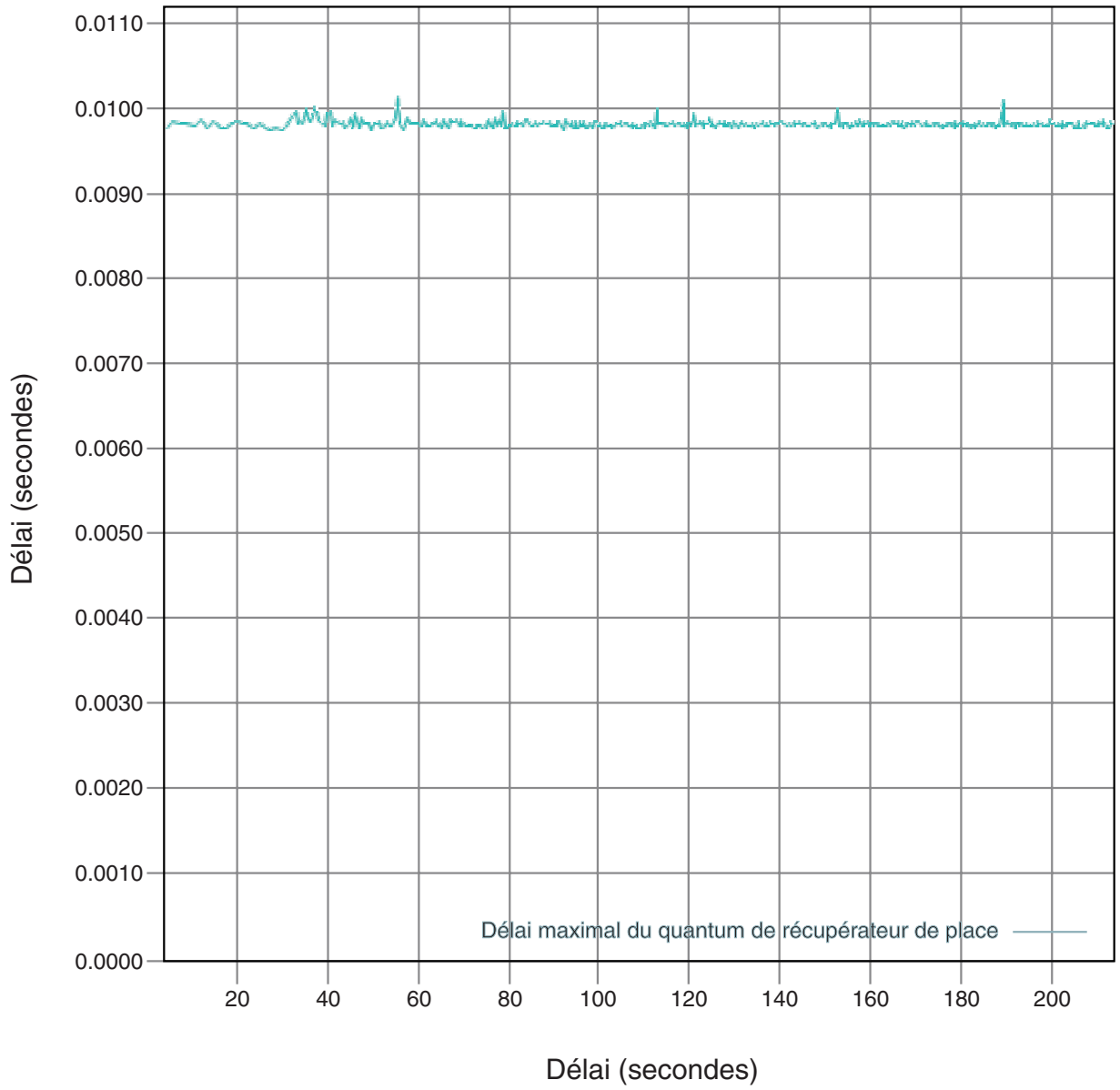

Figure 3. Délais d'interruption réels lorsque le délai d'interruption cible est défini à 10 millisecondes

Si l'on définit le délai d'interruption cible à 15 millisecondes, le graphique des délais d'interruption du récupérateur de place prolixe indique que ces délais ne dépassent pas 15 millisecondes :

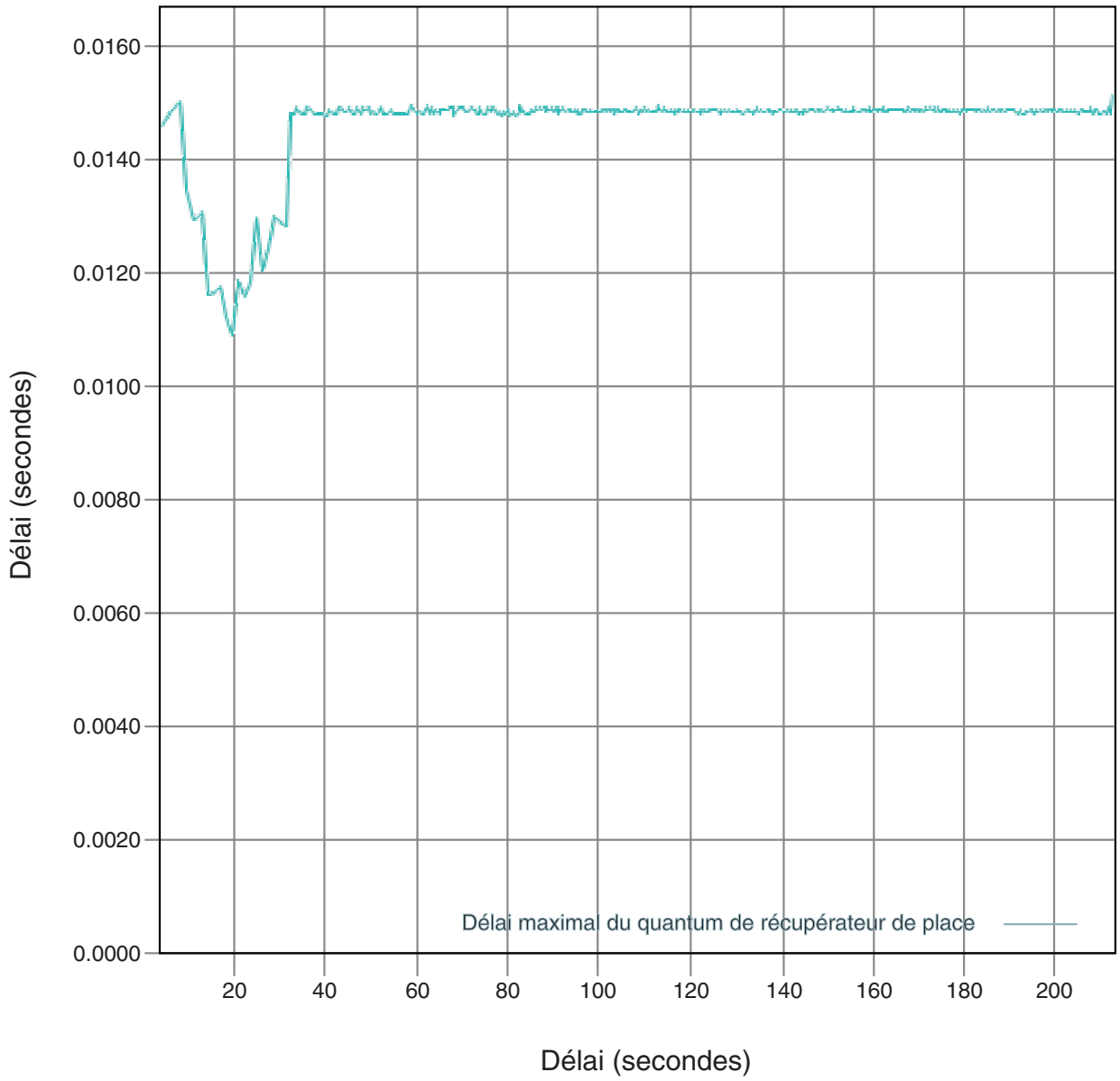

Figure 4. Délais d'interruption réels lorsque le délai d'interruption cible est défini à 15 millisecondes

Contrôle de l'utilisation du processeur

Vous pouvez limiter la quantité de puissance de traitement disponible pour le récupérateur de place Metronome.

Vous pouvez contrôler la récupération de place avec le récupérateur de place Metronome en utilisant l'option **-Xgc:targetUtilization=N** pour limiter la quantité d'UC utilisée par le récupérateur de place.

Par exemple :

```
java -Xgcpolicy:metronome -Xgc:targetUtilization=80 yourApplication
```

Cet exemple spécifie que votre application utilise 80 % de puissance de traitement sur 60 millisecondes. Les 20 % du temps restant sont utilisés pour la récupération de place. Le récupérateur de place Metronome garantit des niveaux d'utilisation dès lors qu'il dispose des ressources suffisantes. La récupération de place commence lorsque la quantité d'espace libre dans le segment de mémoire tombe en dessous d'un seuil défini dynamiquement.

Limitation du récupérateur de place Metronome

Dans certaines circonstances, les interruptions pour récupération de place peuvent être plus longues que prévu.

Au cours de la récupération de place, un processus d'analyse racine est utilisé. Le récupérateur de place parcourt le segment de mémoire en commençant à partir des références actives connues. Ces références incluent :

- Variables de références actives dans les piles d'unité d'exécution actives
- Références statiques

Pour rechercher toutes les références aux objets actifs dans la pile d'une unité d'exécution d'une application, le récupérateur de place parcourt tous les cadres de la pile des appels de l'unité d'exécution. Chaque pile d'unité d'exécution active est analysée dans une étape ininterruptible. Cette analyse doit donc avoir lieu au cours d'une interruption individuelle du récupérateur de place.

Dans ce cas, les performances système peuvent être pires que prévu s'il existe des unités d'exécution très imbriquées dans les piles du fait des longues pauses de récupération de place au début d'un cycle de récupération de place.

Chapitre 6. Développement d'applications

Informations importantes sur l'écriture d'applications en temps réel, y compris les échantillons de code.

- «Exemple de mappe de hachage temps réel»

Exemple de mappe de hachage temps réel

WebSphere Real Time for Linux inclut les implémentations HashMap et HashSet afin d'offrir des performances plus cohérentes pour la méthode put que l'implémentation HashMap standard dans IBM SDK for Java 7.

Le fichier java.util.HashMap standard que fournit IBM fonctionne correctement pour les application à haute capacité de traitement. Il est également utile avec les applications qui connaissent la taille maximale que doit atteindre leur mappe de hachage. Pour les applications qui nécessitent une mappe de hachage qui peut atteindre des tailles variables, en fonction de l'utilisation, il existe un problème potentiel de performances avec le mappage de hachage standard. La mappe de hachage standard est pratique pour y ajouter des entrées en utilisant la méthode put. Toutefois, lorsque la mappe de hachage est pleine, un magasin de sauvegarde plus grand doit être alloué. Cela implique que les entrées du magasin de sauvegarde en cours doivent être migrées. S'il s'agit d'une grande mappe, la durée de l'opération put peut être également longue. Par exemple, l'opération peut prendre plusieurs millisecondes.

WebSphere Real Time for Linux contient un exemple de mappe de hachage temps réel. Il fournit la même interface fonctionnelle que le fichier java.util.HashMap standard, mais il offre des performances plus cohérentes pour la méthode put. Au lieu de créer un magasin de sauvegarde et de migrer toutes les entrées lorsque la mappe de hachage est pleine, l'exemple de mappe de hachage crée un magasin de sauvegarde supplémentaire. Le nouveau magasin de sauvegarde est chaîné aux autres magasins de sauvegarde dans la mappe de hachage. Initialement, le chaînage cause une légère baisse de performance lors de l'affectation du magasin de vidage et de son chaînage aux autres magasins de sauvegarde. Une fois la mappe de hachage de sauvegarde mise à jour, cette opération est plus rapide que de migrer toutes les entrées. L'un des inconvénients de la mappe de hachage temps réel réside dans le fait que les opérations get, put et remove sont légèrement plus lentes, car chaque recherche doit utiliser un groupe de mappes de hachage de sauvegarde et au lieu d'un seul.

Pour tester la mappe de hachage temps réel, ajoutez le fichier `RTTashMap.jar` au début du chemin d'accès aux classes d'amorçage. Si vous avez installé WebSphere Real Time for Linux dans le répertoire `$WRT_ROOT`, ajoutez l'option suivante pour utiliser la mappe de hachage temps réel avec l'application au lieu de la mappe de hachage standard :

```
-Xbootclasspath/p:$WRT_ROOT/demo/realtime/RTHashMap.jar
```

Les fichiers source et classe de l'implémentation de mappe de hachage temps réel sont inclus dans le fichier `demo/realtime/RTHashMap.jar`. En outre, une implémentation java.util.LinkedHashMap et une implémentation java.util.HashSet sont également fournies.

Chapitre 7. Performances

WebSphere Real Time for Linux est optimisé pour des pauses de récupération de données courtes plutôt que pour des performances de capacité de traitement ou une utilisation minimale de la mémoire.

Performances dans les configurations matérielles certifiées

Les systèmes certifiés ont une granularité d'horloge et une vitesse de traitement suffisantes pour atteindre les objectifs de performance WebSphere Real Time for Linux. Par exemple, une application bien écrite exécutée sur un système non surchargé et avec une taille de segment de mémoire appropriée subit généralement des pauses de récupération de place comprises entre 3 environ et 3,2 millisecondes. Au cours des cycles de récupération de place, une application avec les paramètres d'environnement par défaut n'est pas interrompue pendant plus de 30 % du temps écoulé pendant une fenêtre variable de 60 millisecondes. Le temps collectif passé dans les pauses de récupération de place sur une période de 60 millisecondes est normalement égal à 18 millisecondes environ.

Réduction de la variabilité du timing

Les deux sources principales de variabilité dans une machine JVM sont les interruptions de récupération de place. Dans WebSphere Real Time for Linux, les longues pauses potentielles des modes du récupération de place sont éliminées en utilisant le récupérateur de place Metronome. Voir «Utilisation du récupérateur de place Metronome», à la page 15.

Partage de données de classes entre JVM

Le partage des données de classe fournit une méthode transparente permettant de réduire l'encombrement de la mémoire et d'améliorer le temps de démarrage de la machine virtuelle. Pour plus d'informations sur le partage des données de classe, voir «Partage des données de classes entre les machines JVM»

Références compressées

Le récupérateur de place Metronome prend en charge les références compressées et non compressées sur les plateformes 64 bits. Lors de l'utilisation de références compressées, la machine JVM stocke toutes les références aux objets, classes, unités d'exécution et moniteurs sous forme de valeur 32 bits. L'utilisation de références compressées améliore les performances de la plupart des applications, car les objets sont plus petits, ce qui diminue la fréquence de la récupération de place et améliore l'utilisation du cache.

Remarque : La taille du segment de mémoire disponible pour les références compressées est limitée à 28 Go.
Pour plus d'informations sur les références compressées, voir http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.doc.70/diag/understanding/mm_compressed_references.html.

Partage des données de classes entre les machines JVM

Le support des classes partagées est identique avec ou sans l'option **-Xrealtime**.

Vous pouvez partager des données de classe entre les machines virtuelles Java en les stockant dans un fichier cache mappé en mémoire sur disque. Le partage des classes réduit la consommation globale de mémoire virtuelle lorsque plusieurs machines virtuelles partagent un cache. Il diminue également le temps de démarrage d'une machine virtuelle après création du cache. Le cache de classes partagées est indépendant de toute machine virtuelle Java en cours d'exécution et persiste jusqu'à ce qu'il soit supprimé.

Un cache partagé peut contenir les éléments suivants :

- les classes d'amorce
- les classes d'application
- les métadonnées qui décrivent les classes
- le code compilé AOT (Ahead-of-time)

Chapitre 8. Sécurité

Cette section contient des informations importantes relatives à la sécurité.

Considérations de sécurité pour le cache de classes partagées

Le cache de classes partagées vise à faciliter la gestion et l'utilisation du cache, mais la règle de sécurité par défaut peut ne pas convenir.

Lorsque vous utilisez le cache de classes partagées, vous devez connaître les autorisations par défaut des nouveaux fichiers pour pouvoir améliorer la sécurité en limitant l'accès.

Fichier	Autorisations par défaut
Nouveaux caches partagés	Autorisations de lecture pour groupe et autre
Répertoire `javasharedresources`	Lecture, écriture et exécution world

Vous devez disposer de l'autorisation d'écriture sur le fichier de cache et le répertoire cache pour pouvoir détruire ou augmenter un cache.

Modification des autorisations du fichier cache

Pour limiter l'accès à un cache de classes partagées, utilisez la commande **chmod**.

Modification nécessaire	Commande
Limitation de l'accès à l'utilisateur et au groupe	`chmod 770 /tmp/javasharedresources`
Limitation de l'accès à l'utilisateur	`chmod 700 /tmp/javasharedresources`
Limitation de l'utilisateur à la lecture et l'écriture uniquement pour un cache donné	`chmod 600 /tmp/javasharedresources/<file for shared cache>`
Limitation de l'utilisateur et du groupe à la lecture et l'écriture d'un cache donné	`chmod 660 /tmp/javasharedresources/ <ficher de cache partagé>`

Connexion à un cache auquel vous n'êtes pas autorisé à accéder

Si vous tentez de vous connecter à un cache pour lequel vous ne disposez pas des autorisations appropriées, le message d'erreur suivant s'affiche :

```
JVMSHRC226E Error opening shared class cache file
JVMSHRC220E Port layer error code = -302
JVMSHRC221E Platform error message: Permission denied
JVMJ9VM015W Initialization error for library j9shr25(11):
JVMJ9VM009E J9VMDllMain failed
Could not create the Java virtual machine.
```

Chapitre 9. Identification et résolution des problèmes et assistance

Identification et résolution des problèmes pour WebSphere Real Time for Linux

Méthodes générales d'identification des problèmes

L'identification des problèmes permet de comprendre le type d'erreur et de déterminer les mesures à prendre.

Après avoir identifié le problème, vous pouvez exécuter une ou plusieurs des tâches suivantes :

- Résolution du problème
- Recherche d'une solution palliative efficace
- Collecte des données nécessaires pour générer un rapport de bogues à l'attention d'IBM.

Détermination des problèmes Linux

Cette section explique comment identifier les problèmes sous Linux.

Le guide d'utilisation d'IBM SDK for Java 7 fournit des informations utiles sur l'identification et la résolution des problèmes liés à Linux :

- Configuration et vérification de l'environnement Linux
- Techniques générales de débogage
- Diagnostic des pannes
- Débogage des blocages
- Débogage des fuites de mémoire
- Débogage des problèmes de performances

Vous trouverez ces informations ici : IBM SDK for Java 7 - Linux problem determination.

Informations supplémentaires fournies pour IBM WebSphere Real Time for Linux

Configuration et vérification de l'environnement Linux

Dans IBM WebSphere Real Time for Linux, vérifiez que la machine JVM est correctement configurée pour générer un vidage système.

Vidages système Linux (fichiers core)

Lorsqu'une panne se produit, les principales données de diagnostic à obtenir sont le vidage système Linux (fichier core). Pour pouvoir générer ce fichier, vous devez vérifier les paramètres de votre système d'exploitation et l'espace disque disponible comme indiqué dans le guide d'utilisation d'IBM SDK for Java 7.

Paramètres de la machine virtuelle Java

La machine virtuelle Java doit être configurée pour générer des fichiers

core lorsqu'une panne se produit. Exécutez `java -Xdump:what` dans la ligne de commande. La sortie de cette option est :

```
-Xdump:system:
    events=gpf+abort+traceassert+corruptcache,
    label=/mysdk/sdk/jre/bin/core.%Y%m%d.%H%M%S.%pid.dmp,
    range=1..0,
    priority=999,
    request=serial
```

Les valeurs affichées sont les paramètres par défaut. `events=gpf` au minimum doit être défini pour générer un fichier core lorsqu'un blocage se produit. Vous pouvez changer et définir les options avec l'option de ligne de commande **`-Xdump:system[:name1=value1,name2=value2 ...]`**

Techniques générales de débogage

Du fait que les noms d'unité d'exécution Java sont visibles dans le système d'exploitation vous pouvez vous aider de la commande ps pour le débogage. Avec les outils de trace vous devez utiliser les commandes correctes pour IBM WebSphere Real Time for Linux.

Examen des informations de processus

Sortie de la commande ps dans IBM WebSphere Real Time for Linux :

```
ps -eLo pid,tid,rtprio,comm,cmd
13654 13654      - java            jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13655      - main            jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13656      - Signal Reporter jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13661      - JIT Compilation jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13662      - JIT Sampler     jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13666      - Signal Dispatch jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13667      - Finalizer maste jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13668      - Gc Slave Thread jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13669      - Gc Slave Thread jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13670      - Gc Slave Thread jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13671      - Gc Slave Thread jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13672      - Metronome GC Al jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13673      - Thread-2        jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13698      - process reaper  jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13700      - stdout reader j jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13701      - stderr reader j jre/bin/java -Xgcpolicy:metronome -jar example.jar
```

e Sélectionne tous les processus.

L Affiche les unités d'exécution.

o Fournit un format de colonnes prédéfini à afficher. Les colonnes indiquées sont l'ID de processus, l'ID d'unité d'exécution, les règles de planification, la priorité d'unité d'exécution en temps réel, et la commande associée au processus. Ces informations sont utiles pour comprendre quelles sont les unités d'exécution de votre application et de la machine virtuelle s'exécutent en même temps.

Outils de traçage

Les trois outils de traçage sous Linux sont **`strace`**, **`ltrace`** et **`mtrace`**. La commande `man strace` affiche l'ensemble complet des options disponibles.

`strace`
L'outil strace trace les appels système. Vous pouvez l'utiliser sur un processus déjà disponible ou le démarrer avec un nouveau processus. strace enregistre les appels système émis par un programme et les signaux reçus par un processus.

Pour chaque appel système, le nom les argument et la valeur de retour son utilisés. strace vous permet de tracer un programme sans demander la source (aucune recompilation requise). Si vous utilisez strace avec l'option **-f**, il tracera les processus enfant qui ont été créés suite à un appel système affecté. Vous pouvez utiliser strace pour examiner les problèmes de modules d'extension ou essayer de comprendre pourquoi les programmes ne démarrent pas correctement.

Pour utiliser strace avec une application Java, tapez `strace java -Xgcpolicy:metronome <nom-classe>`.

Vous pouvez diriger la sortie de trace depuis l'outil strace vers un fichier à l'aide de l'option `-o`.

ltrace
L'outil ltrace est dépendant de la distribution. Il est très similaire à strace. Cet outil intercepte et enregistre les appels de bibliothèque dynamiques appelés par le processus d'exécution. strace fait de même pour les signaux reçus par le processus d'exécution.

Pour utiliser ltrace avec une application Java, tapez `ltrace java -Xgcpolicy:metronome <nom-classe>`

mtrace
mtrace est fourni avec l'ensemble d'outils de GNU. Il installe les gestionnaires spéciaux de malloc, realloc, et free, et permet de tracer et d'enregistrer toutes les utilisations de ces fonctions dans un fichier. Ce traçage réduit l'efficacité du programme et ne doit pas être activé durant une utilisation normale. Pour utiliser mtrace, définissez **IBM_MALLOCTRACE** sur 1, et **MALLOC_TRACE** pour pointer vers un fichier valide dans lequel seront stockées les informations de traçage. Vous devez disposer d'un accès en écriture sur ce fichier.

Pour utiliser mtrace avec une application Java, tapez :

```
export IBM_MALLOCTRACE=1
export MALLOC_TRACE=/tmp/file
java -Xgcpolicy:metronome <nom-classe>
mtrace /tmp/file
```

Diagnostic des pannes

Suivez les instructions ci-dessous pour collecter les informations relatives aux processus en cours d'exécution et à l'environnement Java avant une panne.

Collecte des informations de processus

Pour rechercher les événements survenus avant la panne, utilisez la commande **gdb** et **bt** pour afficher la trace de pile de l'unité d'exécution défaillante au lieu d'analyser le fichier core.

En savoir plus sur l'environnement Java

Utilisez le vidage Java pour déterminer ce que chaque unité d'exécution effectuait et quelles méthodes Java étaient en cours d'exécution. Mettez les adresses de fonction en correspondance avec les adresses de bibliothèque afin de déterminer la source du code exécuté aux différents points.

Utilisez l'option **-verbose:gc** pour consulter l'état du segment de mémoire Java. Vous devez répondre aux questions suivantes :

- Une pénurie ayant pu provoquer la panne de mémoire s'est-elle produite dans une des zones de mémoire ?

- La panne s'est-elle produite pendant la récupération de place, ce qui indique une éventuelle erreur de récupération de place ?
- La panne s'est-elle produite après la récupération de place, ce qui indique une éventuelle altération de la mémoire ?

Débogage des problèmes de performances

Lors du débogage des problèmes de performance vous devez prendre en considération les éléments spécifiques à IBM WebSphere Real Time for Linux en plus des rubriques du guide d'utilisation d'IBM SDK for Java 7.

Définition de la taille des zones de mémoire

Le segment de mémoire Java est l'un des plus paramètres de réglage les plus importants de votre machine virtuelle Java. Choisissez la taille correcte pour optimiser les performances. L'utilisation de la bonne taille permet au récupérateur de place de fournit l'utilisation requise.

Pour savoir comment faire varier la taille des zones de mémoire, voir «Identification et résolution des incidents du récupérateur de place Metronome», à la page 56.

Compilation et performances JIT

Lorsque vous utilisez la compilations JIT vous devez tenir compte des implications sur le comportement en temps réel.

Restrictions connues de Linux

Linux a fait l'objet d'un développement rapide et de nombreux problèmes se sont posés concernant le l'interaction de la machine virtuelle Java et du système d'exploitation, en particulier dans le domaine des unités d'exécution.

Les restrictions suivantes peuvent affecter votre système Linux.

Unités d'exécution en tant que processus

Si le nombre d'unités d'exécution Java dépasse le nombre maximal de processus autorisés, le programme peut alors :

- Recevoir un message d'erreur
- Recevoir une erreur **SIGSEGV**
- Arrêter

Pour plus d'informations, voir *The Volano Report* sur le site http://www.volano.com/report/index.html.

Restrictions des piles flottantes

Si vous faites une exécution sans piles flottantes, quelle que soit la valeur que vous ayez définie pour **-Xss**, une taille minimum de pile native de 256 Ko pour chaque unité d'exécution est fournie.

Sur un système Linux de pile flottante, les valeurs **-Xss** sont utilisées. Si vous migrez à partir d'un système Linux de pile non flottante, assurez-vous que les valeurs **-Xss** sont assez importantes et ne dépendent pas d'un minimum de 256 Ko.

restrictions glibc

Si vous recevez un message indiquant que le bibliothèque `libjava.so` n'a pas pu être chargée en raison d'un symbole introuvable (tel que __bzero), il est possible que vous ayez installé une version plus ancienne de la bibliothèque d'exécution GNU C, glibc. Le kit de développement de logiciels (SDK) de l'implémentation d'unité d'exécution Linux nécessite glibc version 2.3.2 ou ultérieure.

Restrictions de polices

Lorsque vous effectuez une installation sur un système Red Hat, pour permettre au serveur de polices de trouver les polices Java TrueType, exécutez (par exemple sur Linux IA32) :

```
/usr/sbin/chkfontpath --add /opt/IBM/javawrt3[_64]/jre/lib/fonts
```

Vous devez faire cette opération lors de l'installation et vous devez être connecté en tant que «root» pour exécuter la commande. Pour consulter d'autres problèmes liés aux polices, voir le document *Linux (SDK) et le Runtime Environment User Guide*.

Linux Completely Fair Scheduler affecte les performances Java

Les applications Java qui utilisent la synchronisation intensivement peuvent ne pas fonctionner correctement sur les distributions Linux qui contiennent Completely Fair Scheduler. Completely Fair Scheduler (CFS) est un planificateur adopté dans le noyau Linux principal depuis l'édition 2.6.23. L'algorithme CFS est différent des algorithmes de planification pour les éditions Linux précédentes. Il peut changer les propriétés de performances de certaines applications. En particulier, CFS implémente sched_yield() différemment en permettant à une longue unité d'exécution d'obtenir du temps UC.

Si ce problème survient, le temps UC de l'application Java peut être élevé et le traitement des blocs synchronisés peut être lent. L'application peut sembler arrêtée du fait de la lenteur.

Il existe deux solutions palliatives :

- Démarrez la machine JVM en ajoutant l'argument **-Xthr:minimizeUserCPU**.
- Configurez le noyau Linux pour utiliser une implémentation de sched_yield() qui est plus compatible avec les versions antérieures. Pour ce faire, définissez la propriété du noyau sched_compat_yield sur **1**. Par exemple :

  ```
  echo "1" > /proc/sys/kernel/sched_compat_yield
  ```

N'utilisez pas ces solutions si les performances ne sont pas médiocres.

Ce problème peut affecter le kit de développement et l'environnement d'exécution IBM pour Linux 5.0 (toutes les versions) et 6.0 (toutes les versions jusqu'à SR 4 inclus) exécuté sur les noyaux Linux contenant Completely Fair Scheduler. Pour le kit de développement IBM et l'environnement d'exécution pour Linux version 6.0 après SR 4, l'utilisation de CFS dans le noyau est détectée et l'option **-Xthr:minimizeUserCPU** est activée automatiquement. Parmi les distributions de Linux qui contiennent Completely Fair Scheduler figurent Ubuntu 8.04 et SUSE Linux Enterprise Server 11.

Pour plus d'informations sur CFS, voir Multiprocessing with the Completely Fair Scheduler.

Problèmes de performances sur les noyaux Linux Red Hat MRG

Un problème de configuration lié aux noyaux Red Hat MRG peut provoquer un arrêt inattendu des unités d'application lorsque WebSphere Real Time démarre alors que la récupération de place en mode prolixe est activée. Ces pauses ne sont pas signalées dans la sortie GC prolixe, mais peuvent durer plusieurs millisecondes, selon la configuration du réseau. Les machines virtuelles Java démarrés par des outils utilisateurs LDAP définis à distance sont les plus affectées, car le démon de cache du service de nom (`nscd`) n'est pas démarré, d'où des retards réseau. Pour résoudre le problème, démarrez `nscd`. Procédez comme suit pour vérifier l'état du service `nscd` et corriger le problème :

1. Vérifiez que le démon `nscd` s'exécute en tapant la commande :

   ```
   /sbin/service nscd status
   ```

   Si le démon n'est pas en cours d'exécution, vous voyez s'afficher le message suivant :

   ```
   nscd is stopped
   ```

2. En tant que superutilisateur (root), démarrez le service `nscd` avec la commande suivante :

   ```
   /sbin/service nscd start
   ```

3. En tant que superutilisateur (root), modifiez les informations de démarrage du service `nscd` avec la commande suivante :

   ```
   /sbin/chkconfig nscd on
   ```

Le processus `nscd` est maintenant en cours d'exécution, et démarre automatiquement après le réamorçage.

Détermination des problèmes NLS

La machine JVM contient un support intégré pour différents environnements locaux.

Le guide d'utilisation d'IBM SDK for Java 7 fournit des informations utiles sur l'identification et la résolution des problèmes liés à NLS :

- Présentation des polices de caractères
- Utilitaires de polices de caractères
- Problèmes communs liés à NLS et causes possibles

Vous trouverez ces informations ici : IBM SDK for Java 7 - NLS problem determination.

Détermination des problèmes ORB

L'une des premières tâches à exécuter lorsque vous déboguez un problème ORB consiste à déterminer si le problème se situe au niveau du client ou du serveur de l'application répartie. Considérez une session RMI-IIOP standard comme une communication asynchrone entre un client qui demande d'accéder à un objet et un serveur qui le fournit.

Le guide d'utilisation d'IBM SDK for Java 7 fournit des informations utiles sur l'identification et la résolution des problèmes liés à ORB :

- Identification d'un problème lié à ORB
- Interprétation de la trace de pile
- Interprétation des traces ORB

- Problèmes courants
- Service ORB IBM : collecte des données

Vous trouverez ces informations ici : IBM SDK for Java 7 - ORB problem determination.

Informations supplémentaires fournies pour IBM WebSphere Real Time for Linux

Service ORB IBM : collecte des données

Lors de la collecte de la sortie de version Java pour le service, exécutez la commande suivante :

```
java -Xgcpolicy:metronome -version
```

Tests préliminaires

Lorsqu'un problème survient, le service ORB peut générer une exception `org.omg.CORBA.*` incluant :

- La cause de l'événement
- Un code mineur
- Un état d'achèvement

Pour savoir si le service ORB est la cause du problème, vérifiez les éléments suivants :

- La situation peut être reproduite dans une configuration similaire.
- La compilation JIT est désactivée.
- Aucun code compilé AOT n'est utilisé

Les autres actions incluent :

- La désactivation des processeurs supplémentaires
- La désactivation du traitement multitâche simultané (SMT) si cela est possible
- La suppression des dépendances de mémoire avec le client ou le serveur. Le manque de mémoire physique peut ralentir les performances et provoquer des blocages et des pannes. Pour résoudre ces problèmes, vérifiez que vous disposez d'une quantité de mémoire suffisante.
- La vérification des éléments de réseau physique (pare-feu, liaisons de communication, routeurs, serveurs de noms DNS). Ces éléments sont généralement à l'origine des exceptions CORBA COMM_FAILURE. Envoyez une demande ping à votre poste de travail pour effectuer un test.
- Si l'application utilise une base de données, telle que DB2, utilisez le dernier pilote fiable. Par exemple, pour isoler DB2 AppDriver, utilisez Net Driver qui est plus lent et utilise des sockets, mais plus fiable.

Résolution des erreurs OutOfMemory

Traitement des exceptions OutOfMemoryError.

Pour obtenir des informations d'ordre général sur l'identification et la résolution des incidents liés au récupérateur de place Metronome, voir «Identification et résolution des incidents du récupérateur de place Metronome», à la page 56.

Diagnostic des erreurs OutOfMemoryError

Le diagnostic des exceptions OutOfMemoryError dans le récupérateur de place Metronome peut être plus complexe que dans la machine JVM du fait de la nature périodique du récupérateur de place.

En règle générale, une application temps réel nécessite environ 20% de plus d'espace de segment de mémoire qu'une application standard Java.

Par défaut, la machine JVM produit la sortie de diagnostic suivante lorsqu'une erreur OutOfMemoryError non interceptée se produit :

- Cliché du protocole de sous-réseau. Voir «Utilisation des agents de vidage», à la page 37.
- Cliché de tas. Voir «Utilisation de Heapdump», à la page 45.
- Javadump. Voir «Utilisation de Javadump», à la page 40
- Vidage système ; voir «Utilisation des vidages système et de l'afficheur des vidages système», à la page 48.

Les noms de fichier de vidage figurent dans la sortie de la console :

```
JVMDUMP006I Processing dump event "systhrow", detail "java/lang/OutOfMemoryError" - please wait.
JVMDUMP007I JVM Requesting Snap dump using 'Snap.20081017.104217.13161.0001.trc'
JVMDUMP010I Snap dump written to Snap.20081017.104217.13161.0001.trc
JVMDUMP007I JVM Requesting Heap dump using 'heapdump.20081017.104217.13161.0002.phd'
JVMDUMP010I Heap dump written to heapdump.20081017.104217.13161.0002.phd
JVMDUMP007I JVM Requesting Java dump using 'javacore.20081017.104217.13161.0003.txt'
JVMDUMP010I Java dump written to javacore.20081017.104217.13161.0003.txt
JVMDUMP013I Processed dump event "systhrow", detail "java/lang/OutOfMemoryError".
```

La trace Java qui figure dans la sortie de la console et disponible également dans le vidage Java indique l'emplacement de l'erreur OutOfMemoryError dans l'application Java. Le composant de gestion de la mémoire JVM émet un point de trace qui indique la taille, l'adresse de bloc de classe et le nom d'espace mémoire de l'emplacement défaillant. Ce point de trace se trouve dans le cliché du protocole de sous-réseau :

```
<< lines omitted... >>
09:42:17.563258000 *0xf2888e00       j9mm.101  Event       J9AllocateIndexableObject() returning NULL! 80
bytes requested for object of class 0xf1632d80 from memory space 'Metronome' id=0xf288b584
```

Les zones d'ID de point de trace et de données peuvent être différentes de celles indiquées en fonction du type d'objet à allouer. Dans cet exemple, le point de trace indique que l'échec d'allocation a eu lieu lorsque l'application a tenté d'allouer un objet de 33,6 Mo de type `class 0x81312d8` dans le `segment de mémoire Metronome`, le segment de mémoire `id=0x809c5f0`.

Vous pouvez identifier la zone de mémoire affectée en consultant les informations de gestion de la mémoire dans le vidage Java :

```
NULL           ------------------------------------------------------------------
0SECTION       MEMINFO subcomponent dump routine
NULL           ================================
NULL
1STMEMTYPE     Object Memory
NULL           region     start      end        size       name
1STHEAP        0xF288B584 0xF2A1C000 0xF6A1C000 0x04000000 Default
NULL
1STMEMUSAGE    Total memory available: 67108864 (0x04000000)
```

```
1STMEMUSAGE    Total memory in use:    66676824 (0x03F96858)
1STMEMUSAGE    Total memory free:      00432040 (0x000697A8)

<< lines removed for clarity >>
```

Vous pouvez déterminer le type d'objet à allouer en consultant la section des classes dans le vidage Java:

```
NULL           ------------------------------------------------------------
0SECTION       CLASSES subcomponent dump routine
NULL           ================================
<< lines omitted... >>
1CLTEXTCLLOD    ClassLoader loaded classes
2CLTEXTCLLOAD    Loader *System*(0xF182BB80)
<< lines omitted... >>
3CLTEXTCLASS      [C(0xF1632D80)
```

Les informations dans le vidage Java confirme que la tentative d'allocation concerne un tableau de caractères dans le segment de mémoire normale (`ID=0xF288B584`) et que la taille totale allouée du segment de mémoire, indiquée par la ligne `1STHEAP` appropriée est de 67108864 octets décimaux ou 0x04000000 octets hexadécimaux, soit 64 Mo.

Dans cet exemple, l'allocation qui a échoué est grande par rapport à la taille totale du segment de mémoire. Si l'application doit créer des objets de 33 Mo, l'étape suivante consiste à augmenter la taille du segment en utilisant l'option **-Xmx**.

L'allocation qui échoue est généralement petite par rapport à la taille totale du segment du fait des allocations précédentes qui remplissent le segment de mémoire. Dans ce cas, l'étape suivante consiste à utiliser le cliché de tas pour déterminer la quantité de mémoire allouée aux objets existants.

Le cliché de tas est un fichier binaire compressé contenant tous les objets avec leurs classes d'objet, tailles et références. Analysez le cliché de tas en utilisant l'outil Memory Dump Diagnostics for Java (MDD4J) que vous pouvez télécharger depuis IBM Support Assistant (ISA).

En utilisant MDD4J, vous pouvez charger un cliché de tas et recherchez des structures arborescentes d'objets qui semblent consommer de grandes quantités d'espace de segment de mémoire. L'outil fournit des vues des objets dans le segment de mémoire. Par exemple, MDD4J peut afficher une vue qui détaille des fuites vraisemblablement suspectes et qui indique les cinq premiers objets et packages qui contribuent à la taille de segment de mémoire. La sélection de la vue arborescente fournit des informations supplémentaires sur la nature de l'objet conteneur qui fuit.

Gestion de la mémoire par la machine JVM IBM

La machine JVM IBM nécessite de la mémoire pour divers composants, notamment les régions de mémoire des classes, le code compilé, les objets Java, les piles Java et les piles JNI. Certaines de ces régions de mémoire doivent se trouver dans la mémoire contiguë. Les autres régions de mémoire peuvent être segmentées dans des régions plus petites et interconnectées.

Les classes chargées dynamiquement et le code compilé sont stockés dans des régions de mémoire segmentées pour les classes chargées dynamiquement. Les classes sont aussi divisées en régions de mémoire inscriptibles (classes RAM) et régions de mémoire en lecture seule (classes ROM). Lors de l'exécution, les classes ROM et le code AOT du cache de classe sont mappés dans la mémoire, mais pas chargées dans une région de mémoire contiguë lors du démarrage de l'application.

Lorsque les classes sont référencées par l'application, les classes et le code compilé dans le cache de classes sont mappés dans la mémoire. Le composant ROM de la classe est partagé entre plusieurs processus qui référencent la classe. Le composant RAM de la classe est créé dans les régions de mémoire segmentées pour les classes chargées dynamiquement lorsque la classe est référencée pour la première fois par la machine JVM. Le code compilé par AOT pour les méthodes d'une classe dans le cache des classes est copié dans une région de mémoire de code dynamique exécutable, car ce code n'est pas partagé par les processus. Les classes qui ne sont pas chargées depuis le cache des classes sont similaires aux classes en cache, sauf que les informations de classe ROM sont créées dans des régions de mémoire segmentées pour les classes chargées dynamiquement. Le code généré dynamiquement est stocké dans les mêmes régions de mémoire de code dynamique que celles qui contiennent le code AOT des classes en cache.

La pile de chaque unité d'exécution Java peut couvrir une région de mémoire segmentée. La pile JNI de chaque unité d'exécution occupe une région de mémoire contiguë.

Pour déterminer la manière dont la machine JVM est configurée, exécutez la machine JVM avec l'option **-verbose:sizes**. Cette option affiche les informations sur les régions de mémoire où vous pouvez gérer la taille. Pour les régions de mémoire qui ne sont pas contiguës, un incrément est affiché pour indiquer comment la mémoire est obtenue chaque fois que la taille de la région doit augmenter.

Voici un exemple de sortie en utilisant les options **-Xrealtime -verbose:sizes** :

```
-Xmca32K                    RAM class segment increment
-Xmco128K                   ROM class segment increment
-Xms64M                     initial memory size
-Xmx64M                     memory maximum
-Xmso256K                   operating system thread stack size
-Xiss2K                     java thread stack initial size
-Xssi16K                    java thread stack increment
-Xss256K                    java thread stack maximum size
```

Cet exemple indique que le segment de classe RAM est initialement égal à 0, mais qu'il augmente par bloc de 32 Ko en fonction des besoins. Le segment de classe ROM est initialement égal à 0 et augmente par bloc de 128 Ko en fonction des besoins. Vous pouvez utiliser les options **-Xmca** et **-Xmco** pour contrôler ces tailles. Les segments de classe RAM et ROM augmentent en fonction des besoins et vous n'avez donc pas à changer généralement ces options.

Utilisez l'option **-Xshareclasses** pour déterminer la taille de la région mappée en mémoire si vous utilisez le cache de classes. Voici un exemple de sortie de la commande `java -Xgcpolicy:metronome -Xshareclasses:printStats`.

```
Current statistics for cache "sharedcc_chamlain":


base address = 0xF1BBD000
end address = 0xF2BAF000
allocation pointer = 0xF1CA95A0

cache size = 16776852
free bytes = 15499564
ROMClass bytes = 1198572
AOT bytes = 0
Data bytes = 57300
Metadata bytes = 21416
Metadata % used  = 1%
```

```
# ROMClasses = 368
# AOT Methods = 0
# Classpaths = 1
# URLs           = 0
# Tokens = 0
# Stale classes = 0
% Stale classes = 0%

Cache is 7% full
```

Lors de l'exécution, environ 3 Mo des octets AOT et des octets des métadonnées sont copiés vers la région de mémoire segmentée du code dynamique lors du référencement des classes. Les octets de données sont copiés vers la région de mémoire segmentée des classes RAM lors du référencement des classes.

Utilisation des outils de diagnostic

Un certain nombre d'outils de diagnostic sont fournis pour vous aider à identifier et résoudre les problèmes liés à la machine virtuelle Java d'IBM WebSphere Real Time for Linux.

Le kit SDK IBM pour Java 7 fournit des outils de diagnostic pour vous aider à identifier et résoudre les problèmes liés à la machine virtuelle Java d'IBM WebSphere Real Time for Linux. Cette section présente les outils disponibles et fournit des liens vers des informations supplémentaires relatives à leur utilisation.

Lorsque vous utilisez les outils de diagnostic du kit SDK vous devez tenir compte d'un point important. Lorsque vous appelez la machine virtuelle Java en temps réel, utilisez l'option suivante :

```
java  -Xgcpolicy:metronome
```

Cette option doit être utilisée lors de l'exécution des outils de diagnostic pour la machine virtuelle Java en temps réel. Par exemple, pour afficher les agents de vidage enregistrés destinés à la machine virtuelle Java, entrez :

```
java -Xgcpolicy:metronome -Xdump:what
```

Cette section fournit également des informations supplémentaires sur les différences d'utilisation de ces outils avec IBM WebSphere Real Time for Linux, ainsi que des exemples de sortie pour vous aider à identifier et résoudre les problèmes.

Vous trouverez un récapitulatif des informations d'identification et de résolution des problèmes générées par le kit SDK IBM pour Java 7 dans la section du récapitulatif des informations d'identification et de résolution des problèmes.

Utilisation des agents de vidage

Les agents de vidage sont configurés lors de l'initialisation de la machine virtuelle Java. Ils permettent d'utiliser les événements qui se produisent dans la machine virtuelle Java, tels que la récupération de place ou l'arrêt de la machine JVM, pour exécuter des vidages ou lancer un outil externe.

Le guide d'utilisation d'IBM SDK for Java 7 contient des informations utiles sur les agents de vidage :

- Utilisation de l'option **-Xdump**
- Agents de vidage

- Evénements de vidage
- Contrôle avancé des agents de vidage
- Jetons d'agent de vidage
- Agents de vidage par défaut
- Suppression des agents de vidage
- Variables d'environnement des agents de vidage
- Mappage des signaux
- Emplacements par défaut des agents de vidage

Vous trouverez ces informations ici : IBM SDK for Java 7 - Using dump agents.

Informations supplémentaires relatives à IBM WebSphere Real Time for Linux :

Evénements de vidage

Les agents de vidage sont déclenchés par des événements qui se produisent lorsque la machine virtuelle Java est active. Pour IBM WebSphere Real Time for Linux, la valeur par défaut de l'événement slow et égale à 5 millisecondes.

Certains événements sont filtrés pour améliorer la pertinence de la sortie. Voir «Option de filtrage», à la page 39 pour plus d'informations.

Remarque : Les événements de déchargement et d'extension ne se produisent pas actuellement dans WebSphere Real Time. Les classes sont une mémoire permanente et ne peuvent pas être déchargées.

Remarque : Les événements gpf et abort ne peuvent pas déclencher un cliché de tas (request=prepwalk), préparer le segment de mémoire (request=prepwalk) ou compacter le segment de mémoire (request=compact).

Le tableau suivant répertorie les événements disponibles comme déclencheurs d'agent de vidage :

Evénement	Moment de déclenchement	Option de filtrage
gpf	Erreur de protection générale (GPF).	
utilisateur	La machine JVM reçoit le signal SIGQUIT du système d'exploitation.	
abort	La machine JVM reçoit le signal SIGABRT du système d'exploitation.	
vmstart	Démarrage de la machine virtuelle.	
vmstop	Arrêt de la machine virtuelle.	Filtre le code exit. Par exemple, **`filter=#129..#192#-42#255`**.
load	Chargement d'une classe.	Filtre le nom de classe. Par exemple **`filter=java/lang/String`**.
unload	Déchargement d'une classe.	
throw	Envoi d'une exception.	Filtre le nom de classe d'exception. Par exemple, **`filter=java/lang/OutOfMem*`**
catch	Interception d'une exception.	Filtre le nom de classe d'exception. Par exemple **`filter=*Memory*`**.
uncaught	Non-interception d'une exception Java par l'application.	Filtre le nom de classe d'exception. Par exemple, **`filter=*MemoryError`**.

Evénement	Moment de déclenchement	Option de filtrage
systhrow	Exception Java sur le point d'être émise par la machine virtuelle Java. Différent de l'événement 'throw', car l'exception est déclenchée uniquement pour les conditions d'erreur détectées en interne par la machine virtuelle Java.	Filtre le nom de classe d'exception. Par exemple, **filter=java/lang/OutOfMem***
thrstart	Démarrage d'une nouvelle unité d'exécution.	
blocked	Blocage d'une unité d'exécution.	
thrstop	Arrêt d'une unité d'exception.	
fullgc	Démarrage d'un cycle de récupération de place.	
slow	Une unité d'exécution prend plus de 5 ms pour répondre à une demande JVM interne.	Change le délai pour qu'un événement soit considéré lent. Par exemple, **filter=#300ms** se déclenche lorsqu'une unité d'exécution prend plus de 300 ms pour répondre à une demande JVM interne.
allocation	Un objet Java est alloué avec une taille correspondant à la spécification de filtrage définie	Filtre la taille d'objet. Un filtre doit être défini. Par exemple, **filter=#5m** se déclenche sur les objets de plus de 5 Mo. Les plages sont également prises en charge. Par exemple, **filter=#256k..512k** se déclenche sur les objets dont la taille est comprise entre 256 ko et 512 ko.
traceassert	Une erreur interne s'est produite dans la machine JVM	Non applicable.
corruptcache	La machine JVM détecte que le cache de classes partagées est endommagé.	Non applicable.

Option de filtrage

Certains événements JVM se produisent des milliers de fois pendant la durée de vie d'une application. Les agents de vidage peuvent utiliser des filtres et des plages pour éviter de produire un trop grand nombre de vidages.

Caractères génériques

Vous pouvez utiliser un caractère générique dans le filtre d'événement d'exception en plaçant un astérisque uniquement au début ou à la fin du filtre. Les commandes suivantes ne fonctionnent pas, car le second astérisque ne se trouve pas à la fin :

```
-Xdump:java:events=vmstop,filter=*InvalidArgumentException#*.myVirtualMethod
```

Pour que le filtre puisse fonctionner, vous devez utiliser :

```
-Xdump:java:events=vmstop,filter=*InvalidArgumentException#MyApplication.*
```

Chargement de classe et événements d'exception

Vous pouvez filtrer le chargement de classe (load) et les événements d'exception (throw, catch, uncaught, systhrow) en fonction d'un nom de classe Java :

```
-Xdump:java:events=throw,filter=java/lang/OutOfMem*
-Xdump:java:events=throw,filter=*MemoryError
-Xdump:java:events=throw,filter=*Memory*
```

Vous pouvez filtrer les événements d'exception throw, uncaught et systhrow par nom de méthode Java :

```
-Xdump:java:events=throw,filter=ExceptionClassName[#ThrowingClassName.
throwingMethodName[#stackFrameOffset]]
```

Les parties facultatives sont indiquées entre crochets.

Vous pouvez filtrer les événements d'exception par nom de méthode Java :

```
-Xdump:java:events=catch,filter=ExceptionClassName
[#CatchingClassName.catchingMethodName]
```

Les parties facultatives sont indiquées entre crochets.

Evénement vmstop

Vous pouvez filtrer l'événement d'arrêt de la machine virtuelle Java en utilisant un ou plusieurs codes exit :

```
-Xdump:java:events=vmstop,filter=#129..192#-42#255
```

Evénement slow

Vous pouvez filtrer l'événement slow pour modifier le seuil de délai par défaut égal 5 ms :

```
-Xdump:java:events=slow,filter=#300ms
```

Vous ne pouvez pas affecter au filtre un délai inférieur au délai par défaut.

Evénement d'allocation

Vous devez filtrer l'événement d'allocation pour définir la taille des objets qui provoquent un déclenchement. Vous pouvez définir une taille de filtre comprise entre zéro et la valeur maximale d'un pointeur 32 bits sur les plateformes 32 bits ou la valeur maximale d'un pointeur 64 bits sur les plateformes 64 bits. La définition d'une valeur de filtre inférieure à zéro déclenche un vidage de toutes les allocations.

Par exemple, pour déclencher des vidages sur des allocations d'une taille supérieure à 5 Mo, utilisez :

```
-Xdump:stack:events=allocation,filter=#5m
```

Pour déclencher des vidages sur des allocations dont la taille est comprise entre 256 ko et 512 ko, utilisez :

```
-Xdump:stack:events=allocation,filter=#256k..512k
```

Autres événements

Si vous appliquez un filtre à un événement qui ne prend pas en charge le filtrage, le filtre est ignoré.

Utilisation de Javadump

Javadump génère des fichiers qui contiennent des informations de diagnostic détaillées associées à la machine virtuelle Java et une application Java capturée à un moment donné au cours de l'exécution. Par exemple, les informations peuvent porter sur le système d'exploitation, l'environnement de l'application, les unités d'exécution, les piles, les verrous et la mémoire.

Le guide d'utilisation d'IBM SDK for Java 7 contient des informations utiles sur les Javadumps :

- Activation d'un vidage Java
- Déclenchement d'un vidage Java
- Interprétation d'un vidage Java
- Variables d'environnement et Javadump

Vous trouverez ces informations ici : IBM SDK for Java 7 - Using Javadump.

Des informations supplémentaires et un exemple de sortie relatifs à IBM WebSphere Real Time for Linux sont fournis dans les rubriques suivantes.

Gestion de la mémoire (MEMINFO)

La section `MEMINFO` fournit des informations sur le gestionnaire de mémoire, notamment sur les zones de segment de mémoire, de mémoire pérenne et de mémoire sectorisée.

La section `MEMINFO` d'un fichier de vidage Javadump fournit des informations sur le gestionnaire de mémoire. Voir Utilisation du récupérateur de place métronome pour plus d'informations sur le fonctionnement du composant du gestionnaire de mémoire.

Cette partie du fichier de vidage Javadump fournit plusieurs valeurs de gestion suivantes :

- Quantité de mémoire disponible
- Quantité de mémoire utilisée
- Taille actuelle du segment de mémoire
- Taille actuelle des zones de mémoire pérenne
- Taille actuelle des zones de mémoire sectorisée

Cette section contient également les données d'historique de récupération de place. Ces données sont fournies sous la forme d'une séquence de points de trace horodatés, le dernier point de trace apparaissant en premier.

Les vidages Java produits par la machine virtuelle Java standard contiennent une section «GC History». Ces informations ne figurent pas dans les vidages Java générés en utilisant la machine JVM temps réel. Utilisez l'option **`-verbose:gc`** ou la trace d'instantané JVM pour obtenir des informations sur la récupération de place. Pour plus d'informations, voir «Utilisation des informations verbose:gc», à la page 56 et la section relative aux agents de vidage dans le guide d'utilisation d'IBM SDK for Java 7.

Dans un Javadump, les segments sont des blocs de mémoire alloués par l'exécution Java pour les tâches qui utilisent de grandes quantité de mémoire. Exemples de tâches :

- Gestion des caches JIT
- Enregistrement des classes Java

L'exécution Java alloue également une autre mémoire native qui ne figure pas dans la section `MEMINFO`. La mémoire totale utilisée par les segments d'exécution Java ne représente pas nécessairement l'utilisation de la mémoire complète de l'exécution Java. Un segment d'exécution Java est constitué de la structure des données du segment et d'un bloc associé de mémoire native.

L'exemple suivant montre une sortie standard. Toutes les valeurs sont des valeurs hexadécimales. Les en-têtes de colonne dans la section MEMINFO ont la signification suivante :

```
| 0SECTION       MEMINFO subcomponent dump routine
| NULL           ================================
| NULL
| 1STHEAPTYPE    Object Memory
| NULL           id         start      end        size       space/region
| 1STHEAPSPACE   0x00497030     --         --         --     Generational
| 1STHEAPREGION  0x004A24F0 0x02850000 0x05850000 0x03000000 Generational/Tenured Region
| 1STHEAPREGION  0x004A2468 0x05850000 0x06050000 0x00800000 Generational/Nursery Region
| 1STHEAPREGION  0x004A23E0 0x06050000 0x06850000 0x00800000 Generational/Nursery Region
| NULL
| 1STHEAPTOTAL   Total memory:           67108864 (0x04000000)
| 1STHEAPINUSE   Total memory in use:    33973024 (0x02066320)
| 1STHEAPFREE    Total memory free:      33135840 (0x01F99CE0)
| NULL
| 1STSEGTYPE     Internal Memory
| NULL segment start alloc end type size
| 1STSEGMENT     0x073DFC9C 0x0761B090 0x0761B090 0x0762B090 0x01000040 0x00010000
|        (lines removed for clarity)
| 1STSEGMENT     0x00497238 0x004FA220 0x004FA220 0x0050A220 0x00800040 0x00010000
| NULL
| 1STSEGTOTAL    Total memory:           873412 (0x000D53C4)
| 1STSEGINUSE    Total memory in use:         0 (0x00000000)
| 1STSEGFREE     Total memory free:      873412 (0x000D53C4)
| NULL
| 1STSEGTYPE     Class Memory
| NULL segment start alloc end type size
| 1STSEGMENT     0x0731C858 0x0745C098 0x07464098 0x07464098 0x00010040 0x00008000
|         (lines removed for clarity)
| 1STSEGMENT     0x00498470 0x070079C8 0x07026DC0 0x070279C8 0x00020040 0x00020000
| NULL
| 1STSEGTOTAL    Total memory:          2067100 (0x001F8A9C)
| 1STSEGINUSE    Total memory in use:   1839596 (0x001C11EC)
| 1STSEGFREE     Total memory free:      227504 (0x000378B0)
| NULL
| 1STSEGTYPE     JIT Code Cache
| NULL segment start alloc end type size
| 1STSEGMENT     0x004F9168 0x06960000 0x069E0000 0x069E0000 0x00000068 0x00080000
| NULL
| 1STSEGTOTAL    Total memory:           524288 (0x00080000)
| 1STSEGINUSE    Total memory in use:    524288 (0x00080000)
| 1STSEGFREE     Total memory free:           0 (0x00000000)
| NULL
| 1STSEGTYPE     JIT Data Cache
| NULL segment start alloc end type size
| 1STSEGMENT     0x004F92E0 0x06A60038 0x06A6839C 0x06AE0038 0x00000048 0x00080000
| NULL
| 1STSEGTOTAL    Total memory:           524288 (0x00080000)
| 1STSEGINUSE    Total memory in use:     33636 (0x00008364)
| 1STSEGFREE     Total memory free:      490652 (0x00077C9C)
| NULL
| 1STGCHTYPE     GC History
| 3STHSTTYPE     15:18:14:901108829 GMT j9mm.134 -   Allocation failure end: newspace=7356368/8388608
| oldspace=32038168/50331648 loa=3523072/3523072
| 3STHSTTYPE     15:18:14:901104380 GMT j9mm.470 -   Allocation failure cycle end: newspace=7356416/8388608
| oldspace=32038168/50331648 loa=3523072/3523072
| 3STHSTTYPE     15:18:14:901097193 GMT j9mm.65 -   LocalGC end: rememberedsetoverflow=0
| causedrememberedsetoverflow=0 scancacheoverflow=0 failedflipcount=0 failedflipbytes=0 failedtenurecount=0
| failedtenurebytes=0 flipcount=11454 flipbytes=991056 newspace=7356416/8388608 oldspace=32038168/50331648
| loa=3523072/3523072 tenureage=1
| 3STHSTTYPE     15:18:14:901081108 GMT j9mm.140 -   Tilt ratio: 50
| 3STHSTTYPE     15:18:14:893358658 GMT j9mm.64 -   LocalGC start: globalcount=3 scavengecount=24 weakrefs=0
| soft=0 phantom=0 finalizers=0
| 3STHSTTYPE     15:18:14:893354551 GMT j9mm.63 -   Set scavenger backout flag=false
```

```
| 3STHSTTYPE     15:18:14:893348733 GMT j9mm.135 -   Exclusive access: exclusiveaccessms=0.002
| meanexclusiveaccessms=0.002 threads=0 lastthreadtid=0x00495F00 beatenbyotherthread=0
| 3STHSTTYPE     15:18:14:893348391 GMT j9mm.469 -   Allocation failure cycle start: newspace=0/8388608
| oldspace=38199368/50331648 loa=3523072/3523072 requestedbytes=48
| 3STHSTTYPE     15:18:14:893347364 GMT j9mm.133 -   Allocation failure start: newspace=0/8388608
| oldspace=38199368/50331648 loa=3523072/3523072 requestedbytes=48
| 3STHSTTYPE     15:18:14:866523613 GMT j9mm.134 -   Allocation failure end: newspace=2359064/8388608
| oldspace=38199368/50331648 loa=3523072/3523072
| 3STHSTTYPE     15:18:14:866519507 GMT j9mm.470 -   Allocation failure cycle end: newspace=2359296/8388608
| oldspace=38199368/50331648 loa=3523072/3523072
| 3STHSTTYPE     15:18:14:866513004 GMT j9mm.65 -   LocalGC end: rememberedsetoverflow=0
| causedrememberedsetoverflow=0 scancacheoverflow=0 failedflipcount=5056 failedflipbytes=445632
| failedtenurecount=0 failedtenurebytes=0 flipcount=9212 flipbytes=6017148 newspace=2359296/8388608
| oldspace=38199368/50331648 loa=3523072/3523072 tenureage=1
| 3STHSTTYPE     15:18:14:866493839 GMT j9mm.140 -   Tilt ratio: 64
| 3STHSTTYPE     15:18:14:859814852 GMT j9mm.64 -   LocalGC start: globalcount=3 scavengecount=23 weakrefs=0
| soft=0 phantom=0 finalizers=0
| 3STHSTTYPE     15:18:14:859808692 GMT j9mm.63 -   Set scavenger backout flag=false
| 3STHSTTYPE     15:18:14:859801848 GMT j9mm.135 -   Exclusive access: exclusiveaccessms=0.004
| meanexclusiveaccessms=0.004 threads=0 lastthreadtid=0x00495F00 beatenbyotherthread=0
| 3STHSTTYPE     15:18:14:859801163 GMT j9mm.469 -   Allocation failure cycle start: newspace=0/10747904
| oldspace=38985800/50331648 loa=3523072/3523072 requestedbytes=232
| 3STHSTTYPE     15:18:14:859800479 GMT j9mm.133 -   Allocation failure start: newspace=0/10747904
| oldspace=38985800/50331648 loa=3523072/3523072 requestedbytes=232
| 3STHSTTYPE     15:18:14:652219028 GMT j9mm.134 -   Allocation failure end: newspace=2868224/10747904
| oldspace=38985800/50331648 loa=3523072/3523072
| 3STHSTTYPE     15:18:14:650796714 GMT j9mm.470 -   Allocation failure cycle end: newspace=2868224/10747904
| oldspace=38985800/50331648 loa=3523072/3523072
| 3STHSTTYPE     15:18:14:650792607 GMT j9mm.475 -   GlobalGC end: workstackoverflow=0 overflowcount=0
| memory=41854024/61079552
| 3STHSTTYPE     15:18:14:650784052 GMT j9mm.90 -   GlobalGC collect complete
| 3STHSTTYPE     15:18:14:650780971 GMT j9mm.57 -   Sweep end
| 3STHSTTYPE     15:18:14:650611567 GMT j9mm.56 -   Sweep start
| 3STHSTTYPE     15:18:14:650610540 GMT j9mm.55 -   Mark end
| 3STHSTTYPE     15:18:14:645222792 GMT j9mm.54 -   Mark start
| 3STHSTTYPE     15:18:14:645216632 GMT j9mm.474 -   GlobalGC start: globalcount=2
|              (lines removed for clarity)
| NULL
| NULL           ------------------------------------------------------------------------
```

Unités d'exécution et trace de pile (THREADS)

Pour les programmeurs d'applications, l'un des éléments les plus utiles d'un vidage Java est la section THREADS. Cette section répertorie les unités d'exécution Java, les unités d'exécution natives et les traces de pile.

Une unité d'exécution Java est implémentée par une unité d'exécution native du système d'exploitation. Chaque unité d'exécution est représentée par un ensemble de lignes telles que :

```
"main" J9VMThread:0x41D11D00, j9thread_t:0x003C65D8, java/lang/Thread:0x40BD6070, state:CW, prio=5
 (native thread ID:0xA98, native priority:0x5, native policy:UNKNOWN)
 Java callstack:
 at java/lang/Thread.sleep(Native Method)
 at java/lang/Thread.sleep(Thread.java:862)
 at mySleep.main(mySleep.java:31)
```

Les noms d'unité d'exécution Java sont visibles dans le système d'exploitation lorsque vous utilisez la commande **ps**. Pour plus d'informations sur l'utilisation de la commande **ps**, voir «Techniques générales de débogage», à la page 28.

Un vidage Javadump produit à partir d'une unité d'exécution temps réel sans accès au segment de mémoire peut ne pas contenir certaines informations. Si l'objet de

nom d'unité d'exécution n'est pas visible depuis l'unité d'exécution temps réel sans accès au segment de mémoire, le texte «(access error)» apparaît à la place du nom de l'unité d'exécution.

Les propriétés figurant sur la première ligne correspondent au nom de l'unité d'exécution, aux adresses des structures de l'unité d'exécution de la machine virtuelle Java et de l'objet de l'unité d'exécution Java, à l'état de l'unité d'exécution et à la priorité de l'unité d'exécution Java. Sur la deuxième ligne des propriétés figurent l'ID d'unité d'exécution du système d'exploitation natif, la priorité de l'unité d'exécution du système d'exploitation natif et la règle de planification du système d'exploitation natif.

Les noms d'unité d'exécution sont visibles de trois manières :

- Répertoriés dans des fichiers javacore. Toutes les unités d'exécution n'apparaissent pas dans les fichiers javacore.
- Dans des listes d'unités d'exécution générées à partir du système d'exploitation à l'aide de la commande **ps**.
- Via la méthode java.lang.Thread.getName()

Le tableau ci-dessous fournit des informations sur les noms d'unité d'exécution IBM WebSphere Real Time for Linux.

Tableau 2. Noms d'unités d'exécution dans IBM WebSphere Real Time for Linux

Détail de l'unité d'exécution	**Nom de l'unité d'exécution**
Unité d'exécution JVM interne utilisée par le module de récupération de place pour distribuer la finalisation des objets par le biais des unités d'exécution secondaires.	`Méthode principale de finaliseur`
Unité d'exécution d'alarme utilisée par le récupérateur de place.	`Alarme GC`
Unités d'exécution esclaves utilisées pour la récupération de place.	`Esclave GC`
Unité d'exécution JVM interne utilisée par le module de compilation JIT (just-in-time) pour échantillonner l'utilisation des méthodes dans l'application.	`Echantillonneur JIT`
Unité d'exécution utilisée par la machine virtuelle pour gérer les signaux envoyés par l'application en externe ou en interne.	`Indicateur de signaux`

La priorité de l'unité d'exécution Java est associée à une valeur de priorité du système d'exploitation en fonction de la plateforme. Une valeur élevée pour la priorité de l'unité d'exécution Java indique que l'unité d'exécution a une priorité élevée. En d'autres termes, l'unité d'exécution s'exécute plus fréquemment que celles ayant une priorité plus basse.

Les valeurs d'état peuvent être :

- R - Runnable : l'unité d'exécution peut être exécutée lorsqu'elle le peut.
- CW - Condition Wait : l'unité d'exécution attend, parce que, par exemple :
  - Un appel sleep() est émis.
  - L'unité d'exécution a été bloquée pour E-S.
  - Une méthode wait() est appelée pour attendre la modification d'un moniteur.

– L'unité d'exécution se synchronise avec une autre avec un appel join().

- S – Suspended : l'unité d'exécution a été suspendue par une autre.
- Z – Zombie : l'unité d'exécution a été arrêtée.
- P – Parked : l'unité d'exécution a été parquée par la nouvelle API de concurrence (java.util.concurrent).
- B – Blocked : l'unité d'exécution attend d'obtenir un verrou détenu par un autre élément.

Si une unité d'exécution est à l'état parked ou blocked, la sortie contient une ligne pour cette unité d'exécution ; elle commence par `3XMTHREADBLOCK` et indique la ressource attendue par l'unité d'exécution et, si possible, l'unité d'exécution à laquelle cette ressource appartient. Pour plus d'informations, voir la rubrique relative aux unités d'exécution de type blocked dans le guide d'utilisation d'IBM SDK for Java 7.

Lorsque vous générez un Javadump pour obtenir des informations de diagnostic, la machine JVM met au repos les unités d'exécution Java avant de produire le javacore. L'état de préparation `exclusive_vm_access` est indiqué dans la ligne `1TIPREPSTATE` de la section `TITLE`.

```
1TIPREPSTATE Prep State: 0x4 (exclusive_vm_access)
```

Les unités d'exécution qui exécutaient du code Java lors du javacore ont l'état CW (Condition Wait).

```
3XMTHREADINFO      "main" J9VMThread:0x41481900, j9thread_t:0x002A54A4,
                   java/lang/Thread:0x004316B8, state:CW, prio=5
3XMTHREADINFO1            (native thread ID:0x904, native priority:0x5, native policy:UNKNOWN)
3XMTHREADINFO3          Java callstack:
4XESTACKTRACE               at java/lang/String.getChars(String.java:667)
4XESTACKTRACE               at java/lang/StringBuilder.append(StringBuilder.java:207)
```

La section javacore `LOCKS` montre que ces unités d'exécution attendent sur une horloge JVM interne.

```
2LKREGMON          Thread public flags mutex lock (0x002A5234): <unowned>
3LKNOTIFYQ            Waiting to be notified:
3LKWAITNOTIFY            "main" (0x41481900)
```

Utilisation de Heapdump

Le Heapdump (cliché des tas) décrit le mécanisme IBM Virtual Machine for Java qui génère un vidage de tous les objets actifs qui se trouvent dans le segment de mémoire Java, à savoir ceux utilisés par l'application Java en cours d'exécution.

Le guide d'utilisation d'IBM SDK for Java 7 contient des informations utiles sur les clichés de tas :

- Obtention des clichés de tas
- Outils de traitement des clichés de tas
- Utilisation de **-Xverbose:gc** pour obtenir des informations sur un segment de mémoire
- Variables d'environnement et cliché de tas
- Format de fichier texte Heapdump (classique)
- Format de fichier PHD (Portable Heap Dump)

Vous trouverez ces informations ici : IBM SDK for Java 7 - Using Heapdump.

Informations supplémentaires relatives à IBM WebSphere Real Time for Linux :

Format de fichier texte Heapdump (classique)

Le cliché de tas (Heapdump) texte ou classique correspond à la liste de toutes les instances d'objets dans le segment de mémoire, y compris le type d'objet, sa taille et les références entre les objets.

Enregistrement d'en-tête

L'enregistrement d'en-tête est un enregistrement qui contient une chaîne d'informations de version.

```
// Version: <chaîne contenant le niveau SDK,
la plateforme et le niveau de version JVM>
```

Exemple :

```
// Version: J2RE 7.0 IBM J9 2.6 Linux x86-32 build 20101016_024574_lHdRSr
```

Enregistrements d'objet

Les enregistrements d'objet sont plusieurs enregistrements, un pour chaque instance d'objet dans le segment de mémoire, fournissant l'adresse, la taille, le type et les références de l'objet.

```
<object address, in hexadecimal> [<length in bytes of object instance, in decimal>]
OBJ <object type> <class block reference, in hexadecimal>
<heap reference, in hexadecimal <heap reference, in hexadecimal>...
```

L'adresse de l'objet et les références de segment de mémoire se trouvent dans le segment de mémoire, mais l'adresse de bloc de classe est en dehors du segment. Toutes les références dans l'instance d'objet sont listées, y compris celles qui ont des valeurs NULL. Le type d'objet est soit un nom de classe contenant le package ou un tableau de primitives soit un type de tableau de classes indiqué par sa signature de type JVM standard (voir «Signatures de type VM Java», à la page 48). Les enregistrements d'objet peuvent également contenir des références de bloc de classe supplémentaires, généralement, dans le cas des instances de classe de réflexion.

Exemples :

Une instance d'objet de 8 octets de longueur de type java/lang/String :

```
0x00436E90 [28] OBJ java/lang/String
```

Une adresse de bloc de classe java/lang/String, suivie d'une référence à une instance de tableau de type caractère :

```
0x415319D8 0x00436EB0
```

Une instance d'objet de 44 octets de longueur de type tableau de caractères :

```
0x00436EB0 [44] OBJ [C
```

Une adresse de bloc de classe de type tableau de caractères :

```
0x41530F20
```

Un objet de type tableau de classe interne java/util/Hashtable Entry :

```
0x004380C0 [108] OBJ [Ljava/util/Hashtable$Entry;
```

Un objet de type classe interne java/util/Hashtable Entry :

```
0x4158CD80 0x00000000 0x00000000 0x00000000 0x00000000 0x00421660 0x004381C0
0x00438130 0x00438160 0x00421618 0x00421690 0x00000000 0x00000000 0x00000000
0x00438178 0x004381A8 0x004381F0 0x00000000 0x004381D8 0x00000000 0x00438190
0x00000000 0x004216A8 0x00000000 0x00438130 [24] OBJ java/util/Hashtable$Entry
```

Une adresse de bloc de classe et références de segment de mémoire, y compris les références null :

```
0x4158CB88 0x004219B8 0x004341F0 0x00000000
```

Enregistrements de classe

Les enregistrements de classe sont plusieurs enregistrements, un pour chaque classe chargée, fournissant l'adresse de bloc de classe, la taille, le type et les références de la classe.

```
<class block address, in hexadecimal> [<length in bytes of class block, in decimal>]
CLS <class type>
<class block reference, in hexadecimal> <class block reference, in hexadecimal>...
<heap reference, in hexadecimal> <heap reference, in hexadecimal>...
```

L'adresse de bloc de classe et les références de bloc de classe se trouvent en dehors du segment de mémoire, mais l'enregistrement de classe peut également contenir des références dans le segment de mémoire, généralement pour les membres de données de classe statiques. Toutes les références dans le bloc de classe sont listées, y compris celles qui ont des valeurs null. Le type de classe est soit un nom de classe contenant le package ou un tableau de primitives soit un type de tableau de classe indiquée par sa signature de type JVM standard (voir «Signatures de type VM Java», à la page 48).

Exemples :

Un bloc de classe de 32 octets pour la classe java/lang/Runnable:

```
0x41532E68 [32] CLS java/lang/Runnable
```

Références à d'autres blocs de classe et références de segment de mémoire, y compris les références null :

```
0x4152F018 0x41532E68 0x00000000 0x00000000 0x00499790
```

Un bloc de classe de 168 octets pour la classe java/lang/Math :

```
0x00000000 0x004206A8 0x00420720 0x00420740 0x00420760 0x00420780 0x004207B0
0x00421208 0x00421270 0x00421290 0x004212B0 0x004213C8 0x00421458 0x00421478
0x00000000 0x41589DE0 0x00000000 0x4158B340 0x00000000 0x00000000 0x00000000
0x4158ACE8 0x00000000 0x4152F018 0x00000000 0x00000000 0x00000000
```

Enregistrement de fin 1

L'enregistrement de fin 1 est un enregistrement contenant des nombres d'enregistrements.

```
// Breakdown - Classes: <class record count, in decimal>
Objects: <object record count, in decimal>
ObjectArrays: <object array record count, in decimal>
PrimitiveArrays: <primitive array record count, in decimal>
```

Exemple :

```
// Breakdown - Classes: 321, Objects: 3718, ObjectArrays: 169,
PrimitiveArrays : 2141
```

Enregistrement de fin 2

L'enregistrement de fin 2 est un enregistrement contenant des totaux.

```
// EOF:  Total 'Objects',Refs(null) :
<total object count, in decimal>,
<total reference count, in decimal>
(,total null reference count, in decimal>)
```

Exemple :

```
// EOF:  Total 'Objects',Refs(null) : 6349,23240(7282)
```

Signatures de type VM Java

Les signatures de type VM Java sont des abréviations des types Java, comme indiqué dans le tableau suivant :

Signatures de type VM Java	Type Java
Z	Booléen
S	Octet
G	Caractère
S	Court
I	Entier
J	Long
F	Flottant
D	Double
L <*classe qualifiée complète*> ;	<*classe qualifiée complète*>
[<*type*>	<*type*>[ ] (array of <*type*>)
(<*arg-types*>) <*ret-type*>	méthode

Utilisation des vidages système et de l'afficheur des vidages système

La machine virtuelle Java peut générer des vidages système natifs, appelés également cliché de processus, dans des cas configurables. les vidages système sont généralement volumineux. La plupart des outils utilisés pour analyser ces systèmes sont également spécifiques de la plateforme.Utilisez l'outil **gdb** pour analyser un vidage système sous Linux.

Le guide d'utilisation d'IBM SDK for Java 7 contient des informations utiles sur l'utilisation des vidages système et de l'afficheur des vidages systèmes :

- Présentation des vidages système
- Valeurs par défaut de vidage système
- Utilisation de l'afficheur des vidages
  - Utilisation de **jextract**
  - Problèmes à résoudre avec l'afficheur des vidages
  - Commandes disponibles dans **jdmpview**
  - Exemple de session
  - Référence rapide des commandes **jdmpview**

Vous trouverez ces informations ici : IBM SDK for Java 7 - Using system dumps and the dump viewer.

Informations supplémentaires relatives à IBM WebSphere Real Time for Linux :

Commandes disponibles dans `jdmpview`

`jdmpview` est un outil de ligne de commande interactif qui permet d'explorer les informations d'un vidage système JVM et d'exécuter diverses fonctions d'analyse.

`info jitm`
Affiche les méthodes compilées AOT et JIT et leurs adresses :
- Nom et signature de la méthode
- Adresse de début de la méthode
- Adresse de fin de la méthode

Pour toutes les autres options de commande, voir le guide d'utilisation d'IBM SDK for Java 7.

Traçage des applications Java et la machine virtuelle Java

La trace JVM est une fonction de trace fournie dans IBM WebSphere Real Time for Linux qui a un impact limité sur les performances. Dans la plupart des cas, les données de trace ont un format binaire compressé qui peut être formaté avec le formateur Java fourni.

Le traçage et activé par défaut avec un petit groupe de points de trace placés dans des mémoires tampon. Vous pouvez activer les points de trace lors de l'exécution en utilisant des niveaux, des composants, des noms de groupe ou des identificateurs de point de trace individuels.

Le guide d'utilisation IBM SDK for Java 7 contient des informations détaillées sur le traçage des applications :
- Eléments qui peuvent être tracés
- Types de points de trace
- Traçage par défaut
- Enregistrement des données de trace
- Contrôle de la trace
- Traçage des applications Java
- Traçage des méthodes Java

Lors du traçage d'IBM WebSphere Real Time for Linux vous devez appeler correctement la machine virtuelle Java en temps réel lorsque vous incluez les options de trace. Par exemple, lorsque vous indiquez des options de trace, tapez :

```
java -Xgcpolicy:metronome -Xtrace:<options>
```

Pour les informations relatives à IBM SDK for Java 7, voir : Trace des applications Java et de la machine JVM.

Détermination des problèmes JIT et AOT

Utilisez les options de ligne de commande pour déterminer les problèmes des compilateurs JIT et AOT et optimiser les performances.

Bien qu'IBM WebSphere Real Time for Linux partage certains composants avec IBM SDK for Java 7, JIT et AOT n'ont pas le même comportement. Cette section traite de l'identification et de la résolution des problèmes pour JIT et AOT sur IBM WebSphere Real Time for Linux.

Diagnostic d'un problème JIT ou AOT

Parfois, la compilation de bytecodes valides peut générer du code natif non valide et une erreur du programme Java. En déterminant si le compilateur JIT ou AOT est à l'origine du problème, et si c'est le cas, l'*emplacement* de l'erreur, vous fournissez des informations précieuses au service Java.

Pourquoi et quand exécuter cette tâche

Pour déterminer les méthodes compilées lorsque le cache des classes partagée est rempli, utilisez l'option **-Xaot:verbose** sur la ligne de commande admincache. Par exemple :

```
admincache -Xrealtime -Xaot:verbose -populate -aot my.jar -cp <My Class Path>
```

Cette section explique comment déterminer si le problème est lié au compilateur. Cette section propose également des solutions de contournement et des techniques de débogage pour résoudre les problèmes liés au compilateur.

Désactivation du compilateur JIT ou AOT :

Si vous pensez qu'un problème se produit dans le compilateur JIT ou AOT, désactivez la compilation pour déterminer si le problème persiste. Si le problème apparaît, vous en déduisez que le compilateur n'est pas fautif.

Pourquoi et quand exécuter cette tâche

Le compilateur JIT est actif par défaut. Le compilateur AOT est également activé, mais il ne l'est pas si les classes partagées n'ont pas été activées. Pour plus d'efficacité, les méthodes d'une application Java ne sont pas toutes compilées. La machine JVM gère un nombre d'appels pour chaque méthode dans l'application ; chaque fois qu'une méthode est appelée et interprétée, le nombre d'appels de la méthodes augmente. Lorsque le nombre atteint le seuil de compilation, la méthode est compilée et exécutée en natif.

Le mécanisme de comptage du nombre d'appels étend la compilation des méthodes à toute la vie de l'application en affectant une priorité élevée aux méthodes fréquemment utilisées. Certaines méthodes peu utilisées peuvent ne jamais être compilées. Par conséquent, lorsqu'un programme Java échoue, le problème peut se situer dans le compilateur JIT ou AOT ou autre part dans la machine JVM.

La première étape de détermination de l'échec consiste à déterminer l'*emplacement* du problème. Pour ce faire, vous devez d'abord exécuter le programme Java en mode d'interprétation pur (à savoir, avec les compilateurs JIT et AOT désactivés).

Procédure

1. Supprimez les options **-Xjit** et **-Xaot** (les paramètres associés) depuis la ligne de commande.
2. Utilisez l'option de ligne de commande **-Xint** pour désactiver les compilateurs JIT et AOT. Pour des raisons de performances, n'utilisez pas l'option **-Xint** dans un environnement de production.

Que faire ensuite

L'exécution du programme Java avec la compilation désactivé a l'une des conséquences suivantes :

- L'échec persiste. Le problème ne se trouve pas dans le compilateur JIT ou AOT. Dans certains cas, le programme peut échouer d'une autre manière, mais le problème n'est pas lié au compilateur.
- Le problème disparaît. Il est fort problème qu'il réside dans le compilateur JIT ou AOT.

  Si vous n'utilisez pas de classes partagées, le compilateur JIT est fautif. Si vous en utilisez, vous devez déterminer le compilateur fautif en exécutant l'application en activant seulement la compilation JIT. Exécutez l'application avec l'option **-Xnoaot** à la place de l'option **-Xint**. La conséquence est la suivante :

  – L'échec persiste. Le problème se trouve dans le compilateur JIT. Vous pouvez également utiliser l'option **-Xnojit** au lieu de l'option **-Xnoaot** pour vérifier que seul le compilateur JIT est fautif.
  – Le problème disparaît. Le problème se trouve dans le compilateur AOT.

Désactivation sélective du compilateur JIT :

Si l'échec du programme Java provient d'un problème du compilateur JIT, vous pouvez essayer mieux cerner le problème.

Pourquoi et quand exécuter cette tâche

Par défaut, le compilateur JIT optimise les méthodes à différents niveaux d'optimisation, c'est-à-dire que différentes sélections d'optimisation sont appliquées à différentes méthodes en fonction des nombres d'appels. Les méthodes appelées plus fréquemment sont optimisées à des niveaux supérieurs. En changeant les paramètres du compilateur JIT, vous pouvez contrôler le niveau d'optimisation des méthodes et déterminer si l'optimiseur est fautif et l'optimisation problématique dans ce cas.

Vous pouvez définir les paramètres JIT sous la forme d'une liste séparées des virgules ajoutée à l'option **-Xjit**. La syntaxe est **-Xjit**:<*param1*>,<*param2*>=<*value*>. Par exemple :

```
java -Xjit:verbose,optLevel=noOpt HelloWorld
```

exécute le programme HelloWorld, active la sortie prolixe de JIT et fait que JIT génère du code natif JIT sans exécuter des optimisations.

Procédez comme suit pour déterminer la partie du compilateur qui génère l'erreur :

Procédure

1. Définissez le paramètre JIT **count=0** pour paramétrer le seuil de compilation sur la valeur 0. Ainsi, ce paramètre provoque la compilation de chaque méthode Java avant son exécution. Utilisez **count=0** uniquement pour diagnostiquer les problèmes, car un nombre très important de méthodes sont compilées, notamment des méthodes rarement utilisées. La compilation supplémentaire utilise davantage de ressources de traitement et ralentit votre application. Avec **count=0**, l'application doit échouer immédiatement lorsque la zone fautive est atteinte. Dans certains cas, l'utilisation de **count=1** peut reproduire l'échec de manière plus fiable.

2. Ajoutez **disableInlining** aux paramètres du compilateur JIT. **disableInlining** désactive la génération d'un code plus important et plus complexe. Si le problème disparaît, utilisez **-Xjit:disableInlining** comme solution palliative pendant que le service Java analyse et résout le problème de compilation.
3. Diminuez les niveaux d'optimisation en ajoutant le paramètre **optLevel** et réexécutez le programme jusqu'à ce que le problème disparaisse ou vous atteigniez le niveau «noOpt». Pour un problème de compilation JIT, démarrez avec «scorching» et descendez dans la liste. Les niveaux d'optimisation sont en ordre décroissant :
   a. scorching
   b. veryHot
   c. hot
   d. warm
   e. cold
   f. noOpt

Que faire ensuite

Si l'un de ces paramètres fait disparaître le problème, vous disposez d'une solution de contournement que vous pouvez utiliser. Cette solution est provisoire et peut être utilisée pendant que le service Java analyse et résout le problème de compilation. Si la suppression de **disableInlining** de la liste des paramètres JIT empêche le problème de réapparaître, supprimez-le pour améliorer les performances. Suivez les instructions dans «Recherche de la méthode défaillante» pour améliorer les performances de la solution de contournement.

Si le problème persiste au niveau de l'optimisation «noOpt» une solution palliative consiste à désactiver le compilateur JIT.

Recherche de la méthode défaillante :

Lorsque vous avez déterminé le niveau d'optimisation le plus bas auquel le compilateur JIT ou AOT doit compiler les méthodes pour déclencher l'échec, vous pouvez identifier la partie du programme Java, qui lorsqu'elle est compilée, génère l'échec. Vous pouvez ensuite indiquer au compilateur de limiter la solution de contournement à une méthode, une classe ou un package pour permettre au compilateur de compiler le reste du programme normalement. Pour les échecs du compilateur JIT, si l'échec se produit avec **-Xjit:optLevel=noOpt**, vous pouvez indiquer au compilateur de ne pas compiler la méthode ou les méthodes qui génèrent l'échec.

Avant de commencer

Si une sortie d'erreur similaire à celle ci-dessous s'affiche, vous pouvez l'utiliser pour identifier la méthode défaillante :

```
Unhandled exception
Type=Segmentation error vmState=0x00000000
Target=2_30_20050520_01866_BHdSMr (Linux 2.4.21-27.0.2.EL)
CPU=s390x (2 logical CPUs) (0x7b6a8000 RAM)
J9Generic_Signal_Number=00000004 Signal_Number=0000000b Error_Value=4148bf20 Signal_Code=00000001
Handler1=00000100002ADB14 Handler2=00000100002F480C InaccessibleAddress=0000000000000000
gpr0=0000000000000006 gpr1=0000000000000006 gpr2=0000000000000000 gpr3=0000000000000006
gpr4=0000000000000001 gpr5=0000000080056808 gpr6=0000010002BCCA20 gpr7=0000000000000000
......
Compiled_method=java/security/AccessController.toArrayOfProtectionDomains([Ljava/lang/Object;
Ljava/security/AccessControlContext;)[Ljava/security/ProtectionDomain;
```

Les lignes importantes sont les suivantes :

vmState=0x00000000
Indique que le code erroné ne se trouve pas dans le code d'exécution JVM.

Module= ou Module_base_address=
Ne figure pas dans la sortie (peut être vide ou affecté de la valeur 0), car le code a été compilé par le compilateur JIT et en dehors de la DLL ou bibliothèque.

Compiled_method=
Indique la méthode Java pour laquelle le code compilé a été généré.

Pourquoi et quand exécuter cette tâche

Si la sortie n'indique pas la méthode défaillante, suivez ces étapes pour l'identifier :

Procédure

1. Exécutez le programme Java en ajoutant les paramètres JIT **verbose** et **vlog=**<*filename*> à l'option **-Xjit** ou **-Xaot**. Avec ces paramètres, le compilateur liste les méthodes compilées dans le fichier journal <*filename*>.<*date*>.<*time*>.<*pid*> également appelé *fichier de limites*. Un fichier de limites contient des lignes qui correspondent aux méthodes compilées, telles que :

   ```
   + (hot) java/lang/Math.max(II)I @ 0x10C11DA4-0x10C11DDD
   ```

   Les lignes qui ne commencent pas par le signe Plus sont ignorées par le compilateur dans les étapes suivantes et vous pouvez les supprimer du fichier. Les méthodes compilées par le compilateur AOT commencent par `+ (AOT cold)`. Les méthodes pour lesquelles du code AOT est chargé depuis le cache des classes partagées commencent par `+ (AOT load)`.
2. Exécutez de nouveau le programme avec le paramètre JIT ou AOT **limitFile=**(<*nom fichier*>,<*m*>,<*n*>), où <*nom fichier*> est le chemin du fichier de limites et <*m*> et <*n*> sont des numéros de ligne indiquant la première et la dernière méthodes du fichier de limites à compiler. Le compilateur compile uniquement les méthodes listées sur les lignes <*m*> à <*n*> dans le fichier de limites. Les méthodes qui ne figurent pas dans le fichier de limites et les méthodes figurant sur les lignes en dehors de la plage ne sont pas compilées et aucun code AOT du cache des données partagées de ces méthodes n'est chargé. Si le programme ne génère plus d'erreur, cela implique qu'une ou plusieurs méthodes que vous avez supprimées dans la dernière itération est probablement à l'origine de l'erreur.
3. Facultatif : Si vous déterminez un problème AOT, exécutez le programme une seconde fois avec les mêmes options pour permettre aux méthodes compilées d'être chargées depuis le cache des données partagées. Vous pouvez également ajouter l'option **–Xaot:scount=0** pour que les méthodes compilées AOT stockées dans le cache des données partagées soient utilisées lorsqu'elles sont appelées pour la première fois. Certaines erreurs de compilation AOT se produisent uniquement lorsque le code compilé AOT est chargé depuis le cache des données partagées. Pour déterminer l'origine de ces problèmes, utilisez l'option **–Xaot:scount=0** pour que les méthodes compilées AOT stockées dans le cache des données partagées soient utilisées lorsqu'elles sont appelées pour la première fois, ce qui peut faciliter le reproduction du problème. Notez que si vous affectez à l'option **scount** la valeur 0, vous forcez le chargement du code AOT et suspendez les unités d'exécution Application qui attendent l'exécution

de la méthode. Ne procédez donc ainsi que pour déterminer les problèmes. Les suspensions peuvent durer plus longtemps avec l'option **–Xaot:scount=0** option.

4. Répétez cette procédure en utilisant des valeurs différentes pour <*m*> et <*n*>, autant de fois que nécessaire, pour rechercher le groupe de méthodes minimum qui doivent être compilées pour déclencher l'échec. En divisant par deux le nombre de lignes sélectionnées chaque fois, vous pouvez exécuter une recherche binaire pour la méthode défaillante. En général, vous pouvez réduire le fichier à une seule ligne.

Que faire ensuite

Après avoir localisé la méthode défaillante, vous pouvez désactiver le compilateur JIT ou AOT pour la méthode défaillante uniquement. Par exemple, si la méthode java/lang/Math.max(II)I génère une erreur du programme lorsqu'elle est compilé par JIT avec **optLevel=hot**, vous pouvez exécuter le programme avec :

```
-Xjit:{java/lang/Math.max(II)I}(optLevel=warm,count=0)
```

pour compiler uniquement la méthode défaillante au niveau d'optimisation «warm», mais compiler toutes les autres méthodes normalement.

Si une méthode échoue lorsqu'elle est compilée par JIT avec «noOpt», vous pouvez l'exclure de la compilation en utilisant le paramètre **exclude**={<*method*>} :

```
-Xjit:exclude={java/lang/Math.max(II)I}
```

Si une méthode provoque l'échec du programme lorsque le code AOT est compilé ou chargé depuis le cache des données partagées, excluez la méthode de la compilation AOT et du chargement AOT en utilisant le paramètre **exclude**={<*method*>} :

```
-Xaot:exclude={java/lang/Math.max(II)I}
```

Les méthodes AOT sont compilées au niveau d'optimisation «cold» uniquement. Le blocage de la compilation ou du chargement AOT est la meilleure de ces méthodes.

Identification des échecs de compilation JIT :

Pour les échecs de compilation JIT, analysez la sortie des erreurs pour déterminer si l'échec se produit lorsque le compilateur JIT tente de compiler une méthode.

Si la machine JVM tombe en panne et si vous constatez que l'échec s'est produit dans la bibliothèque JIT (`libj9jit26.so`), il se peut que le compilateur JIT ait généré une erreur lors de la tentative de compilation d'une méthode.

Si l'erreur suivante s'affiche, utilisez-la pour identifier la méthode ayant échoué :

```
Unhandled exception
Type=Segmentation error vmState=0x00050000
Target=2_30_20051215_04381_BHdSMr (Linux 2.4.21-32.0.1.EL)
CPU=ppc64 (4 logical CPUs) (0xebf4e000 RAM)
J9Generic_Signal_Number=00000004 Signal_Number=0000000b Error_Value=00000000 Signal_Code=00000001
Handler1=0000007FE05645B8 Handler2=0000007FE0615C20
R0=E8D4001870C00001 R1=0000007FF49181E0 R2=0000007FE2FBCEE0 R3=0000007FF4E60D70
R4=E8D4001870C00000 R5=0000007FE2E02D30 R6=0000007FF4C0F188 R7=0000007FE2F8C290
......
Module=/home/test/sdk/jre/bin/libj9jit26.so
```

```
Module_base_address=0000007FE29A6000
......
Method_being_compiled=com/sun/tools/javac/comp/Attr.visitMethodDef(Lcom/sun/tools/javac/tree/
JCTree$JCMethodDecl;)
```

Les lignes importantes sont :

vmState=0x00050000
Indique que le compilateur JIT compile du code. Pour obtenir la liste des numéros de code vmState, voir le tableau des balises de vidage Javadump dans le guide d'utilisation d'IBM SDK for Java 7, http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.70.doc/diag/tools/javadump_tags_info.html.

Module=/home/test/sdk/jre/bin/libj9jit26.so
Indique qu'une erreur s'est produite dans `libj9jit26.so`, le module du compilateur JIT.

Method_being_compiled=
Indique la méthode Java compilée.

Si la sortie n'indique pas la méthode qui échoue, utilisez l'option **verbose** avec les paramètres supplémentaires suivants :

```
-Xjit:verbose={compileStart|compileEnd}
```

Ces paramètres **verbose** indiquent quand le compilateur JIT commence à compiler une méthode et quand il a terminé. Si le compilateur JIT échoue sur une méthode (il démarre la compilation, mais échoue), utilisez le paramètre **exclude** pour l'exclure de la compilation (voir «Recherche de la méthode défaillante», à la page 52). Si l'exclusion de la méthode empêche la panne, vous disposez d'une solution de contournement que vous pouvez utiliser pendant que le support résout le problème.

Performances des applications à courte exécution

Le compilateur JIT IBM est optimisé pour les applications à longue exécution généralement utilisées sur un serveur. Vous pouvez utiliser l'option de ligne de commande **-Xquickstart** pour améliorer les performances des applications à exécution courte, notamment celles dans lesquelles le traitement n'est pas concentré dans un petit nombre de méthodes.

-Xquickstart force le compilateur JIT à utiliser un niveau d'optimisation bas par défaut et à compiler moins de méthodes. L'exécution d'un plus petit nombre de compilations plus rapidement peut accélérer le démarrage des applications. Lorsque le compilateur AOT est actif (les classes partagées et la compilation AOT sont activées), **-Xquickstart** force toutes les méthodes sélectionnées pour la compilation à être compilées par le compilateur AOT, ce qui accélère le démarrage des exécutions suivantes. **-Xquickstart** peut affecter les performances si l'option est utilisée avec des applications à exécution longue qui contiennent des méthodes qui utilisent une grande quantité de ressources de traitement. L'implémentation de **-Xquickstart** peut être modifié dans les prochaines versions.

Vous pouvez également essayer d'améliorer les temps de démarrage en ajustant le seuil JIT (en utilisant trial et error). Voir «Désactivation sélective du compilateur JIT», à la page 51 pour plus d'informations.

Comportement de la machine JVM au cours des périodes d'inactivité

Vous pouvez réduire les cycles UC consommés par une machine JVM inactive en utilisant l'option **-XsamplingExpirationTime** pour désactiver l'unité d'exécution d'échantillonnage JIT.

L'unité d'exécution d'échantillonnage JIT profile l'application active Java pour découvrir les méthodes communément utilisées. L'utilisation de la mémoire et du processeur de l'unité d'exécution d'échantillonnage est négligeable et la fréquence de profilage est automatiquement réduite lorsque la machine JVM est inactive.

Dans certains cas, il peut arriver que vous ne vouliez pas qu'une machine JVM inactive consomme des cycles UC. Pour ce faire, définissez l'option **-XsamplingExpirationTime**<*time*>. Affectez à <*time*> le nombre de secondes correspondant à la durée d'exécution de l'unité d'exécution d'échantillonnage. Utilisez cette option avec précaution, car une fois l'unité d'exécution d'échantillonnage désactivée, vous ne pouvez pas réactiver cette dernière. Autorisez l'unité d'exécution d'échantillonnage à s'exécuter suffisamment longtemps pour identifier les optimisations importantes.

Le collecteur de diagnostics

Le collecteur des diagnostics collecte les fichiers de diagnostic Java d'un événement de problème.

La collecte des fichiers requis par le service IBM peut accélérer le traitement des problèmes signalés. Le guide d'utilisation IBM SDK for Java 7 contient des informations détaillées sur l'utilisation du collecteur de diagnostics.

Vous trouverez ces informations ici : IBM SDK for Java 7 - The Diagnostics Collector.

Diagnostics du récupérateur de place

Cette section explique comment identifier et résoudre les problèmes de récupération de place.

Le guide d'utilisation d'IBM SDK for Java 7 contient des informations utiles sur l'identification et la résolution des problèmes de récupération de place :

- Consignation prolixe de la récupération de place
- Suivi de la récupération de place à l'aide de **-Xtgc**

Vous trouverez ces informations ici : IBM SDK for Java 7 - Garbage Collector diagnostics.

Vous trouverez des informations supplémentaires sur le récupérateur de place Metronome IBM WebSphere Real Time for Linux dans les sections suivantes.

Identification et résolution des incidents du récupérateur de place Metronome

En utilisant des options de ligne de commande, vous pouvez contrôler la fréquence de la récupération de place Metronome, les exceptions de manque de mémoire et le comportement de Metronome dans des appels système explicites.

Utilisation des informations verbose:gc :

Vous pouvez utiliser l'option **-verbose:gc** avec l'option **-Xgc:verboseGCCycleTime=N** pour écrire des informations sur la console concernant l'activité du récupérateur de place Metronome. Les propriétés XML dans la sortie **-verbose:gc** de la machine JMV standard ne sont pas toutes créées ou appliquées dans la sortie de récupérateur de place Metronome.

Utilisez l'option **-verbose:gc** pour afficher la quantité de mémoire minimale, maximale et l'espace libre moyen dans le segment de mémoire. De cette manière, vous pouvez vérifier le niveau d'activité et d'utilisation du segment de mémoire et ajuster les valeurs de manière appropriée. L'option **-verbose:gc** écrit des statistiques Metronome sur la console.

L'option **-Xgc:verboseGCCycleTime=N** contrôle la fréquence d'extraction des informations. Elle détermine le délai en millisecondes de vidage des résumés. La valeur par défaut de N est 1 000 millisecondes. La durée du cycle n'implique pas que le résumé soit vidé précisément à ce moment là, mais lorsque l'événement de récupération de place respecte ce critère de temps. La récupération et l'affichage de ces statistiques peuvent affecter les délais d'interruption du récupérateur de place, qui s'accroissent lorsque N diminue.

Un quantum est une période unique d'activité récupérateur de place Metronome qui génère une interruption ou une pause pour une application.

Exemple de sortie verbose:gc

Entrez :

```
java -Xgcpolicy:metronome -verbose:gc -Xgc:verboseGCCycleTime=N myApplication
```

Lorsque la récupération de place démarre, un événement `trigger start` se produit, suivi de n'importe quel nombre d'événements `heartbeat`, puis d'un événement `trigger end` lorsque le déclenchement est satisfait. Cet exemple montre un cycle de récupération de place déclenché comme sortie verbose:gc :

```
<trigger-start id="25" timestamp="2011-07-12T09:32:04.503" />

<cycle-start id="26" type="global" contextid="26" timestamp="2011-07-12T09:32:04.503" intervalms="984.285" />

<gc-op id="27" type="heartbeat" contextid="26" timestamp="2011-07-12T09:32:05.209">
  <quanta quantumCount="321" quantumType="mark" minTimeMs="0.367" meanTimeMs="0.524" maxTimeMs="1.878"
    maxTimestampMs="598704.070" />
  <exclusiveaccess-info minTimeMs="0.006" meanTimeMs="0.062" maxTimeMs="0.147" />
  <free-mem type="heap" minBytes="99143592" meanBytes="114374153" maxBytes="134182032" />
  <thread-priority maxPriority="11" minPriority="11" />
</gc-op>

<gc-op id="28" type="heartbeat" contextid="26" timestamp="2011-07-12T09:32:05.458">
  <quanta quantumCount="115" quantumType="sweep" minTimeMs="0.430" meanTimeMs="0.471" maxTimeMs="0.511"
    maxTimestampMs="599475.654" />
  <exclusiveaccess-info minTimeMs="0.007" meanTimeMs="0.067" maxTimeMs="0.173" />
  <classunload-info classloadersunloaded=9 classesunloaded=156  />
  <references type="weak" cleared="660" />
  <free-mem type="heap" minBytes="24281568" meanBytes="55456028" maxBytes="87231320" />
  <thread-priority maxPriority="11" minPriority="11" />
</gc-op>

<gc-op id="29" type="syncgc" timems="136.945" contextid="26" timestamp="2011-07-12T09:32:06.046">
  <syncgc-info reason="out of memory" exclusiveaccessTimeMs="0.006" threadPriority="11" />
  <free-mem-delta type="heap" bytesBefore="21290752" bytesAfter="171963656" />
</gc-op>
```

```
<cycle-end id="30" type="global" contextid="26" timestamp="2011-07-12T09:32:06.046" />

<trigger-end id="31" timestamp="2011-07-12T09:32:06.046" />
```

Les types d'événements suivants se produisent :

<trigger-start ...>
Début d'un cycle de récupération de place lorsque la quantité de mémoire utilisée est supérieure au seuil de déclenchement. Le seuil par défaut est 50 % du segment de mémoire. L'attribut `intervalms` est l'intervalle entre l'événement `trigger end` précédent (avec id-1) et cet événement `trigger start`.

<trigger-end ...>
Un cycle de récupération de place a ramené la mémoire utilisée sous le seuil de déclenchement. Si un cycle de récupération de place est terminé et que la mémoire utilisée n'est pas retombée en dessous du seuil de déclenchement un nouveau cycle de récupération de place est démarré avec le même ID de contexte. Pour chaque événement `trigger start` il existe un événement `trigger end` ayant le même ID de contexte. L'attribut `intervalms` attribute est l'intervalle entre l'événement `trigger start` précédent et l'événement `trigger end` en cours. Pendant ce temps, un ou plusieurs cycles de récupération de place auront été exécutés jusqu'à ce que la mémoire utilisée retombe en dessous du seuil de déclenchement.

<gc-op id="28" type="heartbeat"...>
Evénement périodique qui collecte des informations (sur la mémoire et le délai) sur tous les quanta de récupération de place pour la période couverte. Un événement de récupération de place peut se produire uniquement entre une paire d'événements `trigger start` et `trigger end` correspondants alors qu'un cycle de récupération de place est en cours. L'attribut `intervalms` est l'intervalle entre l'événement de récupération de place précédent (avec `id` -1) et cet événement de récupération de place.

<gc-op id="29" type="syncgc"...>
Evénement de récupération de place asynchrone (non déterministe). Voir «Récupérations de place synchrones», à la page 59

Les balises XML dans cet exemple ont la signification suivante :

<quanta ...>
Résumé de la durée de pause de quantum au cours de l'intervalle de signalisation de présence incluant la longueur des pauses en millisecondes.

<free-mem type="heap" ...>
Résumé de la quantité d'espace de segment libre au cours de l'intervalle de signalisation de présence, échantillonnée à la fin de chaque quantum de récupération de place.

<classunload-info classloadersunloaded=9 classesunloaded=156 />
Nombre de chargeurs de classes et de classes déchargées au cours de l'intervalle de signalisation de présence.

<references type="weak" cleared="660 />
Nombre et type des objets de référence Java supprimés lors de l'intervalle de signalisation de présence.

Remarque :

- Si un seul quantum de récupération de place se produit dans l'intervalle entre deux signaux de présence, la mémoire libre est échantillonnée uniquement à la

fin du quantum. Par conséquent, les quantités minimale, maximale et moyenne données dans le résumé de signal de présence sont toutes égales.

- Il se peut que l'intervalle entre deux événements de signal de présence soit sensiblement plus long que le cycle défini si le segment de mémoire n'est pas suffisamment plein pour nécessiter une activité de récupération de place. Par exemple, si le programme nécessite une récupération de place une seule fois toutes les quelques secondes, un signal de présence n'apparaîtra vraisemblablement qu'une seule fois toutes les quelques secondes.
- Il se peut que l'intervalle puisse être sensiblement plus long que le cycle défini, car la récupération de place n'a aucune opération à exécuter dans un segment de mémoire qui n'est pas suffisamment plein pour garantir la récupération de place. Par exemple, si le programme nécessite une récupération de place une seule fois toutes les quelques secondes, un signal de présence n'apparaîtra vraisemblablement qu'une seule fois toutes les quelques secondes.

  Si un événement, tel qu'une récupération de place synchrone ou une modification de priorité se produit, les informations de l'événement et les événements en attente, tels que les signaux de présence, sont produits immédiatement en sortie.
- Si le quantum maximal de récupération de place pour une période donnée est trop grand, réduisez l'utilisation cible à l'aide de l'option **-Xgc:targetUtilization**. Cette action laisse au récupérateur de place plus de temps pour travailler. Vous pouvez également augmenter la taille du segment de mémoire avec l'option **-Xmx**. De même, si l'application peut tolérer des délais plus longs que ceux qui sont signalés, vous pouvez augmenter l'utilisation cible et réduire la taille de segment.
- La sortie peut être redirigée vers un fichier journal au lieu de la console avec l'option **-Xverbosegclog:<file>**. Par exemple **-Xverbosegclog:out** écrit la sortie **-verbose:gc** dans le fichier *out*.
- La priorité indiquée dans `thread-priority` est la priorité d'unité d'exécution du système d'exploitation sous-jacente et non une priorité d'unité d'exécution Java.

Récupérations de place synchrones

Une entrée est également écrite dans le journal **-verbose:gc** lorsqu'une récupération de place synchrone (non déterministe) se produit. Cet événement a trois causes :

- Appel explicite `System.gc()` dans le code.
- La machine JVM à court de mémoire exécute alors une récupération de place synchrone pour éviter une condition OutOfMemoryError.
- La machine JVM se ferme pendant un récupération de place continue. La machine JVM ne peut pas annuler la collection, par conséquent elle la termine de manière synchrone et se ferme.

Exemple d'entrée `System.gc()` :

```
<gc-op id="9" type="syncgc" timems="12.92" contextid="8" timestamp=
"2011-07-12T09:41:40.808">
  <syncgc-info reason="system GC" totalBytesRequested="260" exclusiveaccessTimeMs="0.009"
  threadPriority="11" />
  <free-mem-delta type="heap" bytesBefore="22085440" bytesAfter="136023450" />
  <classunload-info classloadersunloaded="54" classesunloaded="234" />
  <references type="soft" cleared="21" dynamicThreshold="29" maxThreshold="32" />
  <references type="weak" cleared="523" />
  <finalization enqueued="124" />
</gc-op>
```

Exemple d'entrée de récupération de place synchrone résultant de l'arrêt de la machine JVM :

```
<gc-op id="24" type="syncgc" timems="6.439" contextid="19" timestamp="2011-07-12T09:43:14.524">
  <syncgc-info reason="VM shut down" exclusiveaccessTimeMs="0.009" threadPriority="11" />
  <free-mem-delta type="heap" bytesBefore="56182430" bytesAfter="151356238" />
  <classunload-info classloadersunloaded="14" classesunloaded="276" />
  <references type="soft" cleared="154" dynamicThreshold="29" maxThreshold="32" />
  <references type="weak" cleared="53" />
  <finalization enqueued="34" />
</gc-op>
```

Les balises XML et les attributs dans cet exemple ont la signification suivante :

<gc-op id="9" type="syncgc" timems="6.439" ...
Cette ligne indique que le type d'événement est une récupération de place synchrone. L'attribut `timems` indique la durée de la récupération de place synchrone en millisecondes.

<syncgc-info reason="..."/>
Cause de la récupération de place synchrone.

<free-mem-delta.../>
Segment de mémoire Java en octets avant et après la récupération de place synchrone en octets.

<finalization .../>
Nombre d'objets en attente de finalisation

<classunload-info .../>
Nombre de chargeurs de classes et de classes déchargées au cours de l'intervalle de signalisation de présence.

<references type="weak" cleared="53" .../>
Nombre et type des objets de référence Java supprimés lors de l'intervalle de signalisation de présence.

Une récupération de place synchrone suite à un manque de mémoire ou l'arrêt d'une machine virtuelle peut se produire uniquement lorsque le récupérateur de place est actif. Elle doit être précédée d'un événement `trigger start`, mais pas nécessairement immédiatement. Certains événements de signalisation de présence se produisent entre un événement `trigger start` et l'événement `synchgc`. La récupération de place synchrone provoquée par `System.gc()` peut avoir lieu à tout moment.

Suivi des quanta de récupération de place

Les quanta de récupération de place peuvent être contrôlés en activant les points de trace `GlobalGCStart` et `GlobalGCEnd`. Ces points de trace sont produits au début et à la fin de toute l'activité de récupération de place Metronome, y compris des récupérations de place synchrones. La sortie de ces points de trace se présente comme suit :

```
03:44:35.281 0x833cd00 j9mm.52 - GlobalGC start: globalcount=3

03:44:35.284 0x833cd00 j9mm.91 - GlobalGC end: workstackoverflow=0 overflowcount=0
```

Entrée de manque de mémoire

Lorsque le segment de mémoire a atteint sa capacité totale, une entrée est écrite dans le journal **-verbose:gc** avant la génération de l'exception OutOfMemoryError. Exemple de sortie :

```
<out-of-memory id="71" timestamp="2011-07-12T10:21:50.135" memorySpaceName="Metronome"
  memorySpaceAddress="0806DFDC"/>
```

Par défaut, un vidage Java est généré suite à une exception OutOfMemoryError. Ce vidage contient des informations sur les de mémoire utilisées par le programme.

```
NULL
1STSEGTOTAL    Total memory:           4066080 (0x003E0B20)
1STSEGINUSE    Total memory in use:    3919440 (0x003BCE50)
1STSEGFREE     Total memory free:       146640 (0x00023CD0)
```

Comportement du récupération de place Metronome lors d'un manque de mémoire :

Par défaut, le récupérateur de place Metronome déclenche une récupération de place non déterministe illimitée lorsque la machine JVM manque de mémoire. Pour éviter ce comportement non déterministe, utilisez l'option **-Xgc:noSynchronousGCOnOOM** pour générer une erreur OutOfMemoryError lorsque la machine JVM manque de mémoire.

La récupération de place illimitée par défaut jusqu'à ce que toute la place possible soit récupérée est exécutée en une seule fois. La durée de pause nécessaire correspond généralement à un beaucoup plus grand nombre de millisecondes qu'un quantum incrémentiel Metronome normal.

Information associée

Utilisation de -Xverbose:gc pour analyser les récupérations de place synchrones

Comportement du récupérateur de place Garbage dans les appels System.gc() explicites :

Si un cycle de récupération de place est en cours, la récupérateur de place Metronome termine le cycle de manière synchrone lorsque `System.gc()` est appelé. Si aucun cycle de récupération de place n'est en cours, un cycle complet synchrone est exécuté lorsque `System.gc()` est appelé. Utilisez `System.gc()` pour nettoyer le segment de mémoire d'une manière contrôlée. Il s'agit d'une opération non déterministe, car elle exécute une récupération de place complète avant de prendre fin.

Certaines applications appellent du logiciel ayant des appels System.gc() où il n'est pas acceptable de créer ces retards non déterministes. Pour désactiver tous les appels System.gc(), utilisez l'option **-Xdisableexplicitgc**.

La sortie de la récupération de place prolixe d'un appel System.gc() a la raison «Collecte de récupération de place système» et vraisemblablement une longue durée :

```
<gc-op id="9" type="syncgc" timems="6.439" contextid="8" timestamp="2011-07-12T09:41:40.808">
  <syncgc-info reason="VM shut down" exclusiveaccessTimeMs="0.009" threadPriority="11"/>
  <free-mem-delta type="heap" bytesBefore="126082300" bytesAfter="156085440"/>
  <classunload-info classloadersunloaded="14" classesunloaded="276"/>
  <references type="soft" cleared="154" dynamicThreshold="29" maxThreshold="32"/>
  <references type="weak" cleared="53"/>
  <finalization enqueued="34"/>
</gc-op>
```

Diagnostics de classes partagées

La compréhension de la résolution des problèmes qui peuvent se produire facilite l'utilisation du mode Classes partagées.

Le guide d'utilisation d'IBM SDK for Java 7contient des informations utiles sur l'identification et la résolution des problèmes liés aux classes partagées :

- Déploiement des classes partagées
- Gestion de la modification du bytecode d'exécution
- Description des mises à jour dynamiques
- Utilisation de l'API Java Helper
- Description de la sortie du diagnostic des classes partagées
- Résolution des problèmes avec les classes partagées

Vous trouverez ces informations ici : IBM SDK for Java 7 - Shared classes diagnostics.

Utilisation de JVMTI

JVMTI est une interface bidirectionnelle qui permet à la machine virtuelle Java et à un agent natif de communiquer. Elle remplace les interfaces JVMDI et JVMPI.

JVMTI permet aux tiers de développer des outils de débogage, de profilage et de contrôle pour la machine virtuelle Java. L'interface contient des mécanismes permettant à l'agent de signaler à la machine virtuelle Java les types d'informations dont elle a besoin. L'interface permet également de recevoir les notifications appropriées. Plusieurs agents peuvent être attachés à une machine virtuelle Java à tout moment.

Le guide d'utilisation IBM SDK for Java 7 contient des informations détaillées sur JVMTI, notamment une section de référence d'API sur les extensions IBM pour JVMTI.

Vous trouverez ces informations ici : IBM SDK for Java 7 - Using JVMTI.

Utilisation de Diagnostic Tool Framework for Java

DTFJ (Diagnostic Tool Framework for Java) est une API (application programming interface) Java d'IBM qui permet de créer des outils de diagnostic Java. DTFJ fonctionne avec les données d'un vidage système ou un Javadump.

Le guide d'utilisation IBM SDK for Java 7 d> contient des informations détaillées sur l'utilisation de DTFJ. Suivez ce lien : Utilisation de Diagnostic Tool Framework for Java

Utilisation d'IBM Monitoring and Diagnostic Tools for Java - Health Center

IBM Monitoring and Diagnostic Tools for Java - Health Center est un outil de diagnostic permettant de contrôler l'état d'une machine virtuelle Java (JVM) active.

Les informations sur IBM Monitoring and Diagnostic Tools for Java - Health Center sont disponibles sur developerWorks et dans le centre de documentation.

Chapitre 10. Référence

Ces rubriques répertorient les options et les bibliothèques de classes pouvant être utilisées avec WebSphere Real Time for Linux

Options de ligne de commande

Vous pouvez définir des options sur la ligne de commande lorsque vous démarrez Java. Des options par défaut ont été choisies pour une utilisation générale optimale.

Indication des options Java et des propriétés système

Vous pouvez définir les propriétés Java et système de trois manières.

Pourquoi et quand exécuter cette tâche

Vous pouvez définir des options Java et des propriétés système des manières suivantes. Voici ces différents méthodes classées par ordre de préférence :

1. En indiquant l'option ou la propriété sur la ligne de commande. Par exemple :

   ```
   java -Dmysysprop1=tcpip -Dmysysprop2=wait -Xdisablejavadump MyJavaClass
   ```

2. En créant un fichier qui contient les options et en l'indiquant sur la ligne de commande à l'aide de l'option **-Xoptionsfile=**<*nom_fichier*>.

   Dans le fichier d'options, définissez chaque option sur une ligne distincte ; vous pouvez utiliser le caractère '\' comme caractère de continuation si une option doit être répartie sur plusieurs lignes. Utilisez le caractère '#' pour les lignes de commentaires. Vous ne pouvez pas définir **-classpath** dans un fichier d'options. Voici un exemple de fichier d'options :

   ```
   #My options file
   -X<option1>
   -X<option2>=\
   <value1>,\
   <value2>
   -D<sysprop1>=<value1>
   ```

3. En créant une variable d'environnement appelée **IBM_JAVA_OPTIONS** contenant les options. Par exemple :

   ```
   export IBM_JAVA_OPTIONS="-Dmysysprop1=tcpip -Dmysysprop2=wait -Xdisablejavadump"
   ```

La dernière option que vous indiquez sur la ligne de commande est prioritaire sur la première. Par exemple, si vous indiquez les options **-Xint -Xjit myClass**, l'option **-Xjit** est prioritaire sur **-Xint**.

Propriétés système

Les propriétés système sont disponibles pour les applications et elles permettent de fournir des informations sur l'environnement d'exécution.

com.ibm.jvm.realtime
: Cette propriété permet aux applications Java de déterminer si elles s'exécutent dans un environnement WebSphere Real Time for Linux.

  Si l'application s'exécute dans l'environnement IBM WebSphere Real Time for RT Linux et a été démarrée avec l'option **-Xrealtime**, la propriété **com.ibm.jvm.realtime** a la valeur «hard».

Si l'application s'exécute dans l'environnement IBM WebSphere Real Time for RT Linux et qu'elle a été démarrée avec l'option **-Xrealtime**, la propriété **com.ibm.jvm.realtime** n'est pas définie.

Si l'application s'exécute dans l'environnement IBM WebSphere Real Time, la propriété **com.ibm.jvm.realtime** a la valeur «soft».

Options standard

Définitions des options standard.

-agentlib:<*nom_bibliothèque*>**[=**<*options*>**]**
Charge la bibliothèque d'agent native <*nom_bibliothèque*> ; par exemple **-agentlib:hprof**. Pour plus d'informations, spécifiez **-agentlib:jdwp=help** et **-agentlib:hprof=help** sur la ligne de commande.

-agentpath:*nom_bibliothèque***[=**<*options*>**]**
Charge une bibliothèque d'agent native en utilisant le nom de chemin d'accès complet.

-assert Affiche l'aide sur les options d'assertion.

-cp ou -classpath <*répertoires et fichiers .zip ou .jar séparés par* **:** >
Définit le chemin de recherche des classes et ressources d'application. Si **-classpath** et **-cp** ne sont pas utilisés et que la variable **CLASSPATH** n'est pas définie, le chemin d'accès aux classes de l'utilisateur correspond par défaut au répertoire en cours (.).

-D<nom_propriété>=<*valeur*>
Définit une propriété système.

-help ou -?
Affiche un message d'aide.

-javaagent:<*jarpath*>**[=**<*options*>**]**
Charge l'agent de langage de programmation Java. Pour plus d'informations, voir la documentation de l'API java.lang.instrument.

-jre-restrict-search
Inclut les JRE privés de l'utilisateur dans la recherche de version.

-no-jre-restrict-search
Exclut les JRE privés de l'utilisateur dans la recherche de version.

-showversion
Affiche la version du produit et continue la procédure.

-verbose:[*class***,***gc***,***dynload***,***sizes***,***stack***,***jni***]**
Active le mode prolixe.

-verbose:class
Consigne une entrée dans stderr pour chaque classe chargée.

-verbose:gc
Voir «Utilisation des informations verbose:gc», à la page 56.

-verbose:dynload
Fournit des informations détaillées lorsque chaque classe est chargée par la machine virtuelle Java, notamment :
- Le nom de classe et le package
- Pour les fichiers classe qui se trouvaient dans un fichier `.jar`, le nom et le chemin du répertoire du fichier `.jar`
- Informations sur la taille et le délai de chargement de la classe.

Les données sont écrites dans stderr. Voici un exemple de sortie :

```
<Loaded java/lang/String from /myjdk/sdk/jre/lib/i386
/softrealtime/jclSC160/vm.jar>
<Class size 17258; ROM size 21080; debug size 0>
<Read time 27368 usec; Load time 782 usec; Translate time 927 usec>
```

Remarque : Les classes chargées depuis la cache des classes partagées n'apparaissent pas dans la sortie **-verbose:dynload**. Utilisez **-verbose:class** pour obtenir des informations sur ces classes.

-verbose:sizes
Ecrit des informations dans stderr, décrivant la quantité de mémoire utilisée pour les piles et les segments de mémoire dans la machine JVM

-verbose:stack
Ecrit des informations dans stderr décrivant l'utilisation des piles Java et C.

-verbose:jni
Consigne des informations dans stderr décrivant les services JNI appelés par l'application et la machine virtuelle JVM.

-version
Affiche les informations de version pour le mode non-temps réel.

-version:*<valeur>*
Requiert l'exécution de la version indiquée.

-X Affiche l'aide concernant les options non standard.

Options non standard

Les options préfixées par **-X** ne sont pas standard et peuvent être modifiées sans préavis.

Le guide d'utilisation d'IBM SDK for Java 7 fournit des informations détaillées sur les options non standard. Vous trouverez ces informations ici : IBM SDK for Java 7 - Command-line options.

Vous trouverez des informations supplémentaires sur IBM WebSphere Real Time for Linux dans les sections suivantes.

Options du récupérateur de place Metronome

Définitions des options du récupérateur de place Metronome.

-Xgc:synchronousGCOnOOM | -Xgc:nosynchronousGCOnOOM
La récupération de place se déclenche, entre autres, lorsque le segment de mémoire est plein. Dans ce cas, l'option **-Xgc:synchronousGCOnOOM** arrête l'application pendant que la récupération de place supprime les objets inutilisés. En cas de nouveau manque de mémoire, diminuez l'utilisation cible pour donner plus de temps à la récupération de place. La définition de **-Xgc:nosynchronousGCOnOOM** implique que lorsque la mémoire du segment est pleine, l'application s'arrête et émet un message de manque de mémoire. La valeur par défaut est **-Xgc:synchronousGCOnOOM**.

-Xnoclassgc
Désactive la récupération de place pour les classes. Cette option désactive

la récupération de place pour le stockage associé aux classes Java qui ne sont plus utilisées par la machine virtuelle Java. Le comportement par défaut est **-Xnoclassgc**.

-Xgc:targetPauseTime=*N*
Définit le délai d'interruption de la récupération de place, *N* indiquant la durée en millisecondes. Lorsque cette option est indiquée, le récupérateur de place fonctionne avec des pauses qui n'excèdent la valeur indiquée. Si cette option n'est pas indiquée, le délai d'interruption par défaut est égal à 3 millisecondes. Par exemple, si vous définissez **-Xgc:targetPauseTime=20** le récupérateur de place effectuera des pauses individuelles d'une durée maximale de 20 millisecondes pendant les opérations de récupération de place.

-Xgc:targetUtilization=*N*
Définit le niveau N% d'utilisation d'application ; le récupérateur de place tente d'utiliser au plus (100-N)% de chaque période. Les valeurs acceptables sont comprises entre 50-80 %. Les applications avec des taux d'allocation faibles peuvent s'exécuter à 90 %. La valeur par défaut est 70 %.

Cet exemple montre que la taille maximale de la mémoire du segment est de 30 Mo. Le récupérateur de place tente d'utiliser 25 % de chaque période, car l'utilisation cible de l'application est 75 %.

```
java -Xgcpolicy:metronome -Xmx30m -Xgc:targetUtilization=75 Test
```

-Xgc:threads=*N*
Spécifie le nombre d'unités d'exécution de récupération de place à exécuter. La valeur par défaut est le nombre de coeurs de processeur disponibles pour le processus. La valeur maximale que vous pouvez définir correspond au nombre de processeurs disponibles pour le système d'exploitation.

-Xgc:verboseGCCycleTime=N
N est le délai en millisecondes de vidage des informations récapitulatives.

Remarque : La durée du cycle n'implique pas que les informations récapitulatives soient vidées précisément à ce moment là, mais lorsque l'événement de récupération de place respecte ce critère de temps.

-Xmx<*size*>
Spécifie la taille du segment de mémoire Java. Contrairement aux autres stratégies de récupération de place, la récupération de place Metronome temps réel ne prend pas en charge l'extension de segment de mémoire. Il n'existe aucune option de taille initiale ou maximale de segment de mémoire. Vous pouvez définir uniquement la taille maximale de segment de mémoire.

Paramètres par défaut de la machine virtuelle Java

Les paramètres par défaut s'appliquent à la machine JVM temps réel lorsque l'environnement dans lequel elle s'exécute n'est pas modifié. Les paramètres courants sont indiqués comme référence.

Les paramètres par défaut peuvent être modifiés en utilisant des variables d'environnement ou des paramètres de lignes de commande lors du démarrage de la machine JVM. Le tableau répertorie les paramètres JVM courants. La dernière colonne indique comment changer le comportement où les clés suivantes s'appliquent :

- **e** : paramètre contrôlé par la variable d'environnement uniquement
- **c** : paramètre contrôlé par le paramètre de ligne de commande uniquement
- **ec** : paramètre contrôlé par la variable d'environnement et le paramètre de ligne de commande, ce dernier étant prioritaire.

Les informations sont fournies comme référence rapide et ne sont pas exhaustive.

Paramètre JVM	Valeur par défaut	Paramètre affecté par
Vidages Java	Activés	ec
Vidages Java en cas de manque de mémoire	Activés	ec
Clichés de tas	Désactivés	ec
Clichés de tas en cas de manque de mémoire	Activés	ec
Vidages système	Activés	ec
Emplacement de génération des fichiers de vidage	Répertoire en cours	ec
Sortie prolixe	Désactivée	c
Recherche du chemin d'accès aux classes d'amorçage	Désactivée	c
Vérifications JNI	Désactivées	c
Débogage distant	Désactivé	c
Vérifications strictes de conformité	Désactivées	c
Démarrage rapide	Désactivé	c
Serveur d'infos de débogage distant	Désactivé	c
Signalisation réduite	Désactivée	c
Chaînage de gestionnaires de signaux	Activé	c
Chemin d'accès aux classes	Non défini	ec
Partage des données de classes	Désactivé	c
Support d'accessibilité	Activé	e
Compilateur JIT	Activé	ec
Compilateur AOT (AOT n'est pas utilisé par la machine virtuelle Java si les classes partagées ne sont pas activées également)	Activé	c
Options de débogage JIT	Désactivées	c
Taille maximale des polices Java2D avec gras algorithmique	14 points	e
Java2D utilise des bitmaps de rendu dans les polices vectorielles	Activé	e
Rastérisation des polices de type libre Java2D	Activée	e
Utilisation des polices AWT par Java2D	Désactivée	e
Paramètres régionaux par défaut	Aucun	e
Délai d'attente avant le démarrage du plug-in	zero	e
Répertoire temporaire	/tmp	e
Redirection du plug-in	Aucun	e
Commutation IM	Désactivée	e
Modificateurs IM	Désactivés	e
Modèle d'unité d'exécution	S/O	e

Paramètre JVM	Valeur par défaut	Paramètre affecté par
Taille de pile initiale des unités d'exécution Java 32 bits. Syntaxe : **-Xiss<size>**	2 ko	c
Taille de pile maximale des unités d'exécution Java 32 bits. Syntaxe : **-Xss<size>**	256 ko	c
Taille de pile des unités d'exécution de système d'exploitation 32 bits. Utilisez **-Xmso<size>**	256 ko	c
Taille initiale de segment de mémoire. Syntaxe **-Xms<size>**	64 Mo	c
Taille maximale de segment de mémoire Java. Syntaxe : **-Xmx<size>**	La moitié de la mémoire disponible avec un minimum de 16 Mo et un maximum de 512 Mo	c
Utilisation d'intervalle de temps cible pour une application. Le récupérateur de place tente d'utiliser ce qui reste. Utilisez **-Xgc:targetUtilization=<percentage>**.	70 %	c
Nombre d'unités d'exécution de récupération de place à exécuter. Use **-Xgc:threads=<value>**	Nombre de coeurs de processeur disponibles pour le processus.	c
Quantité maximale de mémoire pouvant être allouée aux mémoires sectorisées en mode **-Xrealtime**. Utilisez **-Xgc:scopedMemoryMaximumSize=<size>**.	8 Mo	c
Définit la taille de la zone de mémoire pérenne en mode **-Xrealtime**. Use **-Xgc:immortalMemorySize=<size>**	16 Mo	c

Remarque : La «mémoire disponible» correspond à la quantité de mémoire réelle (physique), ou à la valeur **RLIMIT_AS**, selon celle qui est la moins élevée.

Chapitre 11. Remarques

Le présent document peut contenir des informations ou des références concernant certains produits, logiciels ou services IBM non annoncés dans ce pays. Pour plus de détails, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial IBM. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service puisse être utilisé. Tout autre élément fonctionnellement équivalent peut être utilisé, s'il n'enfreint aucun droit d'IBM. Il est de la responsabilité de l'utilisateur d'évaluer et de vérifier lui-même les installations et applications réalisées avec des produits, logiciels ou services non expressément référencés par IBM.

IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Pour le Canada, veuillez adresser votre courrier à :

IBM Director of Commercial Relations
IBM Canada Ltd.
3600 Steeles Avenue East
Markham, Ontario
L3R 9Z7
Canada

Les informations sur les licences concernant les produits utilisant un jeu de caractères double octet peuvent être obtenues par écrit à l'adresse suivante :

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

Le paragraphe suivant ne s'applique ni au Royaume-Uni, ni dans aucun autre pays dans lequel il serait contraire aux lois locales.

LE PRESENT DOCUMENT EST LIVRE EN L'ETAT. IBM DECLINE TOUTE RESPONSABILITE, EXPLICITE OU IMPLICITE, RELATIVE AUX INFORMATIONS QUI Y SONT CONTENUES, Y COMPRIS EN CE QUI CONCERNE LES GARANTIES DE VALEUR MARCHANDE OU D'ADAPTATION A VOS BESOINS. Certaines juridictions n'autorisent pas l'exclusion des garanties implicites, auquel cas l'exclusion ci-dessus ne vous sera pas applicable.

Le présent document peut contenir des inexactitudes ou des coquilles. Il est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. IBM peut modifier sans préavis les produits et logiciels décrits dans ce document.

Les références à des sites Web non IBM sont fournies à titre d'information uniquement et n'impliquent en aucun cas une adhésion aux données qu'ils contiennent. Les éléments figurant sur ces sites Web ne font pas partie des éléments du présent produit IBM et l'utilisation de ces sites relève de votre seule responsabilité.

IBM pourra utiliser ou diffuser, de toute manière qu'elle jugera appropriée et sans aucune obligation de sa part, tout ou partie des informations qui lui seront fournies.

Les licenciés souhaitant obtenir des informations permettant : (i) l'échange des données entre des logiciels créés de façon indépendante et d'autres logiciels (dont celui-ci), et (ii) l'utilisation mutuelle des données ainsi échangées, doivent adresser leur demande à :

- JIMMAIL@uk.ibm.com [contact Hursley Java Technology Center (JTC)]

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Le logiciel sous licence décrit dans ce document et tous les éléments sous licence disponibles s'y rapportant sont fournis par IBM conformément aux termes du Contrat sur les produits et services IBM, des Conditions internationales d'utilisation des logiciels IBM ou de tout autre accord équivalent.

Les données de performance indiquées dans ce document ont été déterminées dans un environnement contrôlé. Par conséquent, les résultats peuvent varier de manière significative selon l'environnement d'exploitation utilisé. Certaines mesures évaluées sur des systèmes en cours de développement ne sont pas garanties sur tous les systèmes disponibles. En outre, elles peuvent résulter d'extrapolations. Les résultats peuvent donc varier. Il incombe aux utilisateurs de ce document de vérifier si ces données sont applicables à leur environnement d'exploitation.

Les informations concernant des produits non IBM ont été obtenues auprès des fournisseurs de ces produits, par l'intermédiaire d'annonces publiques ou via d'autres sources disponibles. IBM n'a pas testé ces produits et ne peut confirmer l'exactitude de leurs performances ni leur compatibilité. Elle ne peut recevoir aucune réclamation concernant des produits non IBM. Toute question concernant les performances de produits non IBM doit être adressée aux fournisseurs de ces produits.

Marques

IBM, le logo IBM et ibm.com sont des marques d'International Business Machines Corporations aux Etats-Unis et/ou dans certains autres pays. Si ces marques et d'autres marques d'IBM sont accompagnées d'un symbole de marque (® ou ™), ces symboles signalent des marques d'IBM aux Etats-Unis à la date de publication de ce document. Ces marques peuvent également exister et éventuellement avoir été enregistrées dans d'autres pays. La liste actualisée de toutes les marques d'IBM est disponible sur la page Web http://www.ibm.com/legal/copytrade.shtml.

Adobe, le logo Adobe, PostScript et le logo PostScript sont des marques d'Adobe Systems Incorporated aux Etats-Unis et/ou dans dans certains autres pays.

Intel et Itanium sont des marques d'Intel Corporation aux Etats-Unis et/ou dans certains autres pays.

Linux est une marque de Linus Torvalds aux Etats-Unis et/ou dans certains autres pays.

Java ainsi que tous les logos et toutes les marques incluant Java sont des marques d'Oracle et/ou de ses sociétés affiliées.

Les autres noms de sociétés, de produits et de services peuvent appartenir à des tiers.

Remarques

Le présent document peut contenir des informations ou des références concernant certains produits, logiciels ou services IBM non annoncés dans ce pays. Pour plus de détails, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial IBM. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service puisse être utilisé. Tout autre élément fonctionnellement équivalent peut être utilisé, s'il n'enfreint aucun droit d'IBM. Il est de la responsabilité de l'utilisateur d'évaluer et de vérifier lui-même les installations et applications réalisées avec des produits, logiciels ou services non expressément référencés par IBM.

IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Pour le Canada, veuillez adresser votre courrier à :

IBM Director of Commercial Relations
IBM Canada Ltd.
3600 Steeles Avenue East
Markham, Ontario
L3R 9Z7
Canada

Les informations sur les licences concernant les produits utilisant un jeu de caractères double octet peuvent être obtenues par écrit à l'adresse suivante :

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

Le paragraphe suivant ne s'applique ni au Royaume-Uni, ni dans aucun autre pays dans lequel il serait contraire aux lois locales.

LE PRESENT DOCUMENT EST LIVRE EN L'ETAT. IBM DECLINE TOUTE RESPONSABILITE, EXPLICITE OU IMPLICITE, RELATIVE AUX INFORMATIONS QUI Y SONT CONTENUES, Y COMPRIS EN CE QUI CONCERNE LES GARANTIES DE VALEUR MARCHANDE OU D'ADAPTATION A VOS BESOINS. Certaines juridictions n'autorisent pas l'exclusion des garanties implicites, auquel cas l'exclusion ci-dessus ne vous sera pas applicable.

Le présent document peut contenir des inexactitudes ou des coquilles. Il est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. IBM peut modifier sans préavis les produits et logiciels décrits dans ce document.

Les références à des sites Web non IBM sont fournies à titre d'information uniquement et n'impliquent en aucun cas une adhésion aux données qu'ils contiennent. Les éléments figurant sur ces sites Web ne font pas partie des éléments du présent produit IBM et l'utilisation de ces sites relève de votre seule responsabilité.

IBM pourra utiliser ou diffuser, de toute manière qu'elle jugera appropriée et sans aucune obligation de sa part, tout ou partie des informations qui lui seront fournies.

Les licenciés souhaitant obtenir des informations permettant : (i) l'échange des données entre des logiciels créés de façon indépendante et d'autres logiciels (dont celui-ci), et (ii) l'utilisation mutuelle des données ainsi échangées, doivent adresser leur demande à :

- JIMMAIL@uk.ibm.com [contact Hursley Java Technology Center (JTC)]

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Le logiciel sous licence décrit dans ce document et tous les éléments sous licence disponibles s'y rapportant sont fournis par IBM conformément aux termes du Contrat sur les produits et services IBM, des Conditions internationales d'utilisation des logiciels IBM ou de tout autre accord équivalent.

Les données de performance indiquées dans ce document ont été déterminées dans un environnement contrôlé. Par conséquent, les résultats peuvent varier de manière significative selon l'environnement d'exploitation utilisé. Certaines mesures évaluées sur des systèmes en cours de développement ne sont pas garanties sur tous les systèmes disponibles. En outre, elles peuvent résulter d'extrapolations. Les résultats peuvent donc varier. Il incombe aux utilisateurs de ce document de vérifier si ces données sont applicables à leur environnement d'exploitation.

Les informations concernant des produits non IBM ont été obtenues auprès des fournisseurs de ces produits, par l'intermédiaire d'annonces publiques ou via d'autres sources disponibles. IBM n'a pas testé ces produits et ne peut confirmer l'exactitude de leurs performances ni leur compatibilité. Elle ne peut recevoir aucune réclamation concernant des produits non IBM. Toute question concernant les performances de produits non IBM doit être adressée aux fournisseurs de ces produits.

Marques

IBM, le logo IBM et ibm.com sont des marques d'International Business Machines Corporations aux Etats-Unis et/ou dans certains autres pays. Si ces marques et d'autres marques d'IBM sont accompagnées d'un symbole de marque (® ou ™), ces symboles signalent des marques d'IBM aux Etats-Unis à la date de publication de ce document. Ces marques peuvent également exister et éventuellement avoir été enregistrées dans d'autres pays. La liste actualisée de toutes les marques d'IBM est disponible sur la page Web http://www.ibm.com/legal/copytrade.shtml.

Adobe, le logo Adobe, PostScript et le logo PostScript sont des marques d'Adobe Systems Incorporated aux Etats-Unis et/ou dans dans certains autres pays.

Intel et Itanium sont des marques d'Intel Corporation aux Etats-Unis et/ou dans certains autres pays.

Linux est une marque de Linus Torvalds aux Etats-Unis et/ou dans certains autres pays.

Java ainsi que tous les logos et toutes les marques incluant Java sont des marques d'Oracle et/ou de ses sociétés affiliées.

Les autres noms de sociétés, de produits et de services peuvent appartenir à des tiers.

Index

Caractères spéciaux

A

C

D

E

F

G

H

I

J

L

M

Imprimé en France